Naiad: a timely dataflow system
Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, Martín Abadi
Naiad is a distributed system for high-throughput, low-latency, cyclic dataflow
What do we look for in a dataflow system?

- Batch Processors
- Stream Processors
- Graph Processors
What do we look for in a dataflow system?

- Batch Processors
- Stream Processors
- Graph Processors

System
- Consistent
- Low Latency
- Supports Iteration

Implies
Batch Processors (MapReduce)

- Operate on “data at rest”
- “every night, calculate the previous day’s total sales”
- High throughput
- Easy to use and scale (very popular!)
Batch Processors (MapReduce)

• Operate on “data at rest”
• “every night, calculate the previous day’s total sales”
• High throughput
• Easy to use and scale (very popular!)

• High latency
• No support for incremental computation
  • Have to recalculate from scratch every time
Stateless Stream Processing

• Operate on “data in motion”
• “Running sum of total sales”
• Fed timestamped events as they occur by a message broker/queue (Kafka, Debezium, etc)
Stateless Stream Processing

- Operate on “data in motion”
- “Running sum of total sales”
- Fed timestamped events as they occur by a message broker/queue (Kafka, Debezium, etc)

- Out of order arrivals mean aggregations not guaranteed to be correct
Graph Processing

• “Find the degree of connection (shortest path) between me and another user on LinkedIn”
• GraphX (on top of Spark), Giraph
• No clear victor in the space, open problem
• Why? Graph traversals require **iterative algorithms**
Graph Processing

• “Find the degree of connection (shortest path) between me and another user on LinkedIn”

• GraphX (on top of Spark), Giraph

• No clear victor in the space, open problem

• Why? Graph traversals require **iterative algorithms**

• Most dataflow systems are acyclic

• Hard to parallelize iteration
One framework to rule them all?

- Timely Dataflow
- Consistent
- Low Latency
- Supports Iteration
A shared runtime

Batch Processing

Stream Processing

Graph Processing

Timely Dataflow
How does Timely Dataflow achieve all this?
How does Timely Dataflow achieve all this?

Timestamps!
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Input
Map
Filter
Sum
Output

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday’s records until it’s seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Input
- laptop @ yesterday
- crisps @ today
- iphone @ today
- fruit @ yesterday
- ipad @ yesterday

Map

Filter

Sum

Output

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday's records until it's seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Input:
- ipad @ yesterday
- crisps @ today
- iphone @ today
- fruit @ yesterday
- laptop @ yesterday

Map:
- $500 @ yesterday
- $1 @ today
- $800 @ today
- $3 @ yesterday
- $900 @ yesterday

Filter:

Sum:

Output:

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday's records until it's seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Filter</th>
<th>Sum</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipad @ yesterday</td>
<td>$500 @ yesterday</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>crisps @ today</td>
<td>$1 @ today</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iphone @ today</td>
<td>$800 @ today</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fruit @ yesterday</td>
<td>$3 @ yesterday</td>
<td>$800 @ today</td>
<td></td>
<td></td>
</tr>
<tr>
<td>laptop @ yesterday</td>
<td>$900 @ yesterday</td>
<td>$900 @ yesterday</td>
<td>$900 @ yesterday</td>
<td></td>
</tr>
</tbody>
</table>

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday’s records until it’s seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Filter</th>
<th>Sum</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipad @ yesterday</td>
<td>$500 @ yesterday</td>
<td>$500 @ yesterday</td>
<td>$900 @ yesterday</td>
<td></td>
</tr>
<tr>
<td>crisps @ today</td>
<td>$1 @ today</td>
<td>$800 @ today</td>
<td>$800 @ today</td>
<td></td>
</tr>
<tr>
<td>iphone @ today</td>
<td>$800 @ today</td>
<td>$800 @ today</td>
<td>$800 @ today</td>
<td></td>
</tr>
<tr>
<td>fruit @ yesterday</td>
<td>$3 @ yesterday</td>
<td>$3 @ yesterday</td>
<td>$3 @ yesterday</td>
<td></td>
</tr>
<tr>
<td>laptop @ yesterday</td>
<td>$900 @ yesterday</td>
<td>$900 @ yesterday</td>
<td>$900 @ yesterday</td>
<td></td>
</tr>
</tbody>
</table>

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday's records until it's seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Input
- ipad @ yesterday
- crisps @ today
- iphone @ today
- fruit @ yesterday
- laptop @ yesterday

Map
- $500 @ yesterday
- $1 @ today
- $800 @ today
- $3 @ yesterday
- $900 @ yesterday

Filter
- $800 @ today
- $3 @ yesterday

Sum
- $500 @ yesterday
- $800 @ today
- $900 @ yesterday

Output
- $800 @ today
- $900 @ yesterday

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday's records until it's seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Input
- ipad @ yesterday
- crisps @ today
- iphone @ today
- fruit @ yesterday
- laptop @ yesterday

Map
- $500 @ yesterday
- $1 @ today
- $800 @ today
- $3 @ yesterday
- $900 @ yesterday

Filter
- $500 @ yesterday
- $800 @ today
- $900 @ yesterday

Sum

Output

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday's records until it's seen all of them.
Stateless Stream Consistency

How much revenue are we making from high value item sales, per day?

Sum needs to know the minimum timestamp still upstream so that it can statefully hold onto yesterday’s records until it’s seen all of them.
Timely Data Flow Consistency

How much revenue are we making from high value item sales, per day?

Event plane

$500 @ yesterday
$800 @ today
$900 @ yesterday
today
yesterday
yesterday

Progress tracking plane

“Minimum timestamp after me”

Sum maintains internal state
Timely Data Flow Consistency
How much revenue are we making from high value item sales, per day?

Event plane
- $500 @ yesterday
- $800 @ today
- $900 @ yesterday
- yesterday
- today
- yesterday

Progress tracking plane
“Minimum timestamp after me”

Sum maintains internal state

Sum
$900 @ yesterday

Output

$900 @ yesterday
Timely Data Flow Consistency

How much revenue are we making from high value item sales, per day?

Event plane

$500 \ @ \ yesterday$

$800 \ @ \ today$

$900 \ @ \ yesterday$

Progress tracking plane

“Minimum timestamp after me”

Sum maintains internal state
Timely Data Flow Consistency

How much revenue are we making from high value item sales, per day?

Event plane

$500 \at yesterday$
$800 \at today$
$900 \at yesterday$

today
yesterday
yesterday

Progress tracking plane

“Minimum timestamp after me”

Sum maintains internal state
Coordination: The usual way

- Each worker has no awareness of larger graph
- Each operator is stateless (in most systems)
Parallelization: The Timely Dataflow way

- Coordination only occurs where needed (the Sum operator)
- Consistency guaranteed, while maintaining low latency!
Efficiency gains at scale

• Paths don’t coordinate unless they need to!
Recap

Timely Dataflow

- Workers coordinate to determine minimum timestamp upstream at each operator
- Supports Iteration? But only when needed
Recap

Timely Dataflow

Consistent

- Workers coordinate to determine minimum timestamp upstream at each operator
Recap

Timely Dataflow

- Workers coordinate to determine minimum timestamp upstream at each operator
- But only when needed
Recap

Timely Dataflow

Consistent  Low Latency  Supports Iteration?

- Workers coordinate to determine minimum timestamp upstream at each operator
- But only when needed
Expressive iteration

• Timestamps + stateful vertices make iteration achievable

• Append a loop counter to each timestamp on entry to loop

• Increment counter by passing through feedback node

• Arbitrarily nested loops supported (just append more loop counters to the timestamp)

(Materialize.com, 2020)

• Still maintains consistency and low latency!
Expressive iteration

- Timestamps + stateful vertices make iteration achievable
- Append a loop counter to each timestamp on entry to loop
- Increment counter by passing through feedback node
- Arbitrarily nested loops supported (just append more loop counters to the timestamp)
- Still maintains consistency and low latency!
## Performance (SCC)

<table>
<thead>
<tr>
<th></th>
<th>Connected</th>
<th>Cores</th>
<th>Livejournal</th>
<th>orkut</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphX</td>
<td>128</td>
<td>59s</td>
<td>53s</td>
<td></td>
</tr>
<tr>
<td>SociaLite</td>
<td>128</td>
<td>54s</td>
<td>78s</td>
<td></td>
</tr>
<tr>
<td>Myria</td>
<td>128</td>
<td>37s</td>
<td>57s</td>
<td></td>
</tr>
<tr>
<td>BigDatalog</td>
<td>128</td>
<td>27s</td>
<td>33s</td>
<td></td>
</tr>
<tr>
<td>Timely Dataflow</td>
<td>1, 2</td>
<td>20s, 11s</td>
<td>43s, 26s</td>
<td></td>
</tr>
</tbody>
</table>

(Clockworks, 2019)
## Performance (SCC)

<table>
<thead>
<tr>
<th></th>
<th>Connected</th>
<th>Cores</th>
<th>Livejournal</th>
<th>orkut</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GraphX</strong></td>
<td>128</td>
<td>59s</td>
<td>53s</td>
<td></td>
</tr>
<tr>
<td><strong>Socialite</strong></td>
<td>128</td>
<td>54s</td>
<td>78s</td>
<td></td>
</tr>
<tr>
<td><strong>Myria</strong></td>
<td>128</td>
<td>37s</td>
<td>57s</td>
<td></td>
</tr>
<tr>
<td><strong>BigDatalog</strong></td>
<td>128</td>
<td>27s</td>
<td>33s</td>
<td></td>
</tr>
<tr>
<td><strong>Timely Dataflow</strong></td>
<td>1, 2</td>
<td>20s, 11s</td>
<td>43s, 26s</td>
<td></td>
</tr>
<tr>
<td><strong>Differential update</strong></td>
<td>1, 2</td>
<td>98us, 109us</td>
<td>200us, 216us</td>
<td></td>
</tr>
</tbody>
</table>

(Clockworks, 2019)
So why isn’t everyone using it?
So why isn’t everyone using it?

(Opinions are my own)
Generalized to a fault?

• Timely Dataflow is only the “simplest solution” when you need all of these properties (consistency, low latency, iteration)
  
  • Hard to come up with use case: real-time graph analytics?

• For most large-scale data processing, batch solutions are sufficient (and much simpler to use/reason about)
  
  • i.e. LinkedIn only calculates up to 3 degrees of separation, which can be done via batch processing, albeit inefficiently (but who cares??)

• Timely Dataflow’s fault tolerance unclear compared to other frameworks

• Basic API is elegant, but unintuitive
10 years on: who is using it?

• Has been entirely rewritten in Rust over past 5 years

• Timely dataflow by itself is too low level / too complex for most users

• Ability to build abstractions on top of it has become the killer feature

• Frank McSherry is now a founder of materialize.com, “The Streaming Database for Real-time Analytics”

• Users write normal SQL queries, which are automatically translated to Timely Dataflow magic
10 years on: who is using it?

- Has been entirely rewritten in Rust over past 5 years
- Timely dataflow by itself is too low level / too complex for most users
- Ability to build abstractions on top of it has become the killer feature
- Frank McSherry is now a founder of materialize.com, “The Streaming Database for Real-time Analytics”
- **Users write normal SQL queries**, which are automatically translated to Timely Dataflow magic

Materialize raises a $60M Series C, bringing total funding to over $100M

We’re excited to share the news that we have raised $60 million in Series C funding, led by our ne…
Materialize

The Only Platform for Streaming Joins

While other stream processing tools are limited to basic joins, if any, Materialize brings the same powerful join capabilities found in a traditional database to streams of data.

Materialize Join Capabilities:

- **Inner, Left (outer), Right, Full** and **Cross** Joins
- Multi-way Joins
- Lateral joins

```
CREATE MATERIALIZED VIEW user_join AS
    SELECT
        u.id, SUM(p.amount), last_login
    FROM users
    -- Inner join
    JOIN purchases p ON p.user_id = u.id
    -- Left (outer) join + subquery
    LEFT JOIN (  
        SELECT user_id, MAX(ts) as last_login  
        FROM logins GROUP BY 1  
    ) lg ON lg.user_id = u.id
    GROUP BY u.id;
```
In conclusion

• Timely Dataflow is a “shared foundation” for dataflow applications

• Guarantees consistency, low latency, and supports iteration

• A design and engineering feat

However…

• “Killer usecase” is rare

• API is complex, too low-level

  • Materialize, other abstractions address this for specific usecases
Questions?
Recommended Watching

• “Timely Dataflow in three easy steps | Frank McSherry” (https://youtu.be/yOnPmVf4YWo)

• “Naiad: A Timely Dataflow System” (https://youtu.be/yyhMI9r0A9E)

• “It's About Time: An Introduction to Timely Dataflow | Clockworks” (https://youtu.be/ZN7nOwJTSZ0)