Paper Review
Naiad: A Timely Dataflow System¹

Theo Long
19th October 2022

¹All figures and information from Murray et. al (2013), unless otherwise specified
Naiad: High Performance Parallel Processing

Abstract

Naiad is a distributed system for executing data parallel, cyclic dataflow programs. It offers the high throughput of batch processors, the low latency of stream processors, and the ability to perform iterative and incremental computations. Although existing systems offer some of these features, applications that require all three have relied on multiple platforms, at the expense of efficiency, maintainability, and simplicity. Naiad resolves the complexities of combining these features in one framework.

A new computational model, timely dataflow, underlies Naiad and captures opportunities for parallelism across a wide class of algorithms. This model enriches dataflow computation with timestamps that represent logical points in the computation and provide the basis for an efficient, lightweight coordination mechanism.

We show that many powerful high-level programming models can be built on Naiad's low-level primitives, enabling such diverse tasks as streaming data analysis, iterative machine learning, and interactive graph mining. Naiad outperforms specialized systems in their target application domains, and its unique features enable the development of new high-performance applications.

1 Introduction

Many data processing tasks require low-latency interactive access to results, iterative sub-computations, and consistent intermediate outputs so that sub-computations can be nested and composed. Figure 1 exemplifies these requirements: the application performs iterative processing on a real-time data stream, and supports interactive queries on a fresh, consistent view of the results. However, no existing system satisfies all three requirements: stream processors can produce low-latency results for non-iterative algorithms [3, 5, 9, 38], batch systems can iterate synchronously at the expense of latency [27, 30, 43, 45], and trigger-based approaches support iteration with only weak consistency guarantees [29, 36, 46]. While it might be possible to assemble the application in Figure 1 by combining multiple existing systems, applications built on a single platform are typically more efficient, succinct, and maintainable.

Our goal is to develop a general-purpose system that fulfills all of these requirements and supports a wide variety of high-level programming models, while achieving the same performance as a specialized system. To this end, we have developed a new computational model, timely dataflow, that supports the following features:

1. structured loops allowing feedback in the dataflow,
2. stateful dataflow vertices capable of consuming and producing records without global coordination, and
3. notifications for vertices once they have received all records for a given round of input or loop iteration.

439
Naiad: High Performance Parallel Processing

1. High throughput (like batch processing)
1. High throughput (like batch processing)
2. Low latency (like stream processing)
Naiad: High Performance Parallel Processing

1. High throughput (like batch processing)
2. Low latency (like stream processing)
3. Iterative Computation – i.e. loops
Naiad: High Performance Parallel Processing

1. High throughput (like batch processing)
2. Low latency (like stream processing)
3. Iterative Computation – i.e. loops
4. Strong Consistency Guarantees
What is Timely Dataflow?
What is Timely Dataflow?

• Directed Graph of stateful vertices
What is Timely Dataflow?

• Directed Graph of stateful vertices
• Allows for nested cycles
What is Timely Dataflow?

• Directed Graph of stateful vertices
• Allows for nested cycles
• Vertices send and receive timestamped messages allowing for a partial ordering of computations that includes time
What is Timely Dataflow?

• Directed Graph of stateful vertices
• Allows for nested cycles
• Vertices send and receive timestamped messages allowing for a partial ordering of computations that includes time
• This allows for parallel computation across epochs
What is Timely Dataflow?

- Directed Graph of stateful vertices
- Allows for nested cycles
- Vertices send and receive timestamped messages allowing for a partial ordering of computations that includes time
- This allows for parallel computation across epochs
- Global notification when all messages of given timestamp received -> Allows us to reason about result ‘as of’ a certain time
Key Abstraction: Timestamps

Figure 2: The Naiad software stack exposes a low-latency distributed runtime.

The third feature makes it possible to produce consistent libraries, DSLs, and applications can be built.

Figure 4: An example vertex with one input and two outputs, producing the distinct input records on the first output, counting how often each distinct input record was observed and generating a count for each one on the second output.
Achieving Timely Dataflow with Timestamps

• Dataflow through the graph is done by **events** (messages or notification requests between nodes)
Achieving Timely Dataflow with Timestamps

• Dataflow through the graph is done by events (messages or notification requests between nodes)

• Time always increases, so for an event $e$, only events with time $t \geq te$ can depend on $e$
Achieving Timely Dataflow with Timestamps

• Dataflow through the graph is done by events (messages or notification requests between nodes)

• Time always increases, so for an event $e$, only events with time $t \geq t_e$ can depend on $e$

• Scheduler a list of pointstamps = time + location of unprocessed events
  • Each pointstamp has an occurrence count and predecessor count
Achieving Timely Dataflow with Timestamps

• Dataflow through the graph is done by events (messages or notification requests between nodes)

• Time always increases, so for an event $e$, only events with time $t \geq te$ can depend on $e$

• Scheduler a list of pointstamps = time + location of unprocessed events
  • Each pointstamp has an occurrence count and predecessor count

• When a pointstamp no longer has any predecessors (it is on the frontier), all corresponding events can be released
Making Timely Dataflow Distributed
Making Timely Dataflow Distributed

- Naiad implements **data parallelism** – operating on multiple parts of the data at once
Making Timely Dataflow Distributed

• Naiad implements **data parallelism** – operating on multiple parts of the data at once

• Logical computation graph translated into **physical** graph
Making Timely Dataflow Distributed

• Naiad implements **data parallelism** – operating on multiple parts of the data at once
• Logical computation graph translated into **physical** graph
• Dependencies are always measured in the logical graph – this is not always optimal, but guarantees correctness
Making Timely Dataflow Distributed

• Each worker maintains a local view of global occurrence and precedence counts
Making Timely Dataflow Distributed

• Each worker maintains a **local** view of global occurrence and precedence counts
• When an event is dispatched by a worker, it broadcasts update to all other workers
Making Timely Dataflow Distributed

• Each worker maintains a local view of global occurrence and precedence counts
• When an event is dispatched by a worker, it broadcasts update to all other workers
• By processing incoming updates from a given worker in FIFO order, we guarantee that the local frontier is always a subset of the global frontier
Optimizing Broadcasts

1. Progress tracking is done on logical graph, which is much smaller than physical graph

2. Updates are accumulated in a local buffer – updates with the same pointstamp are combined into one update
What do Naiad programs look like?

```csharp
// 1a. Define input stages for the dataflow.
var input = controller.NewInput<string>();

// 1b. Define the timely dataflow graph.
// Here, we use LINQ to implement MapReduce.
var result = input.SelectMany(y => map(y))
    .GroupBy(y => key(y),
        (k, vs) => reduce(k, vs));

// 1c. Define output callbacks for each epoch
result.Subscribe(result => { ... });

// 2. Supply input data to the query.
input.OnNext(/* 1st epoch data */);
input.OnNext(/* 2nd epoch data */);
input.OnNext(/* 3rd epoch data */);
input.OnCompleted();
```
Naiad is Fast for Iterative Graph Computation

- Want to perform connected components on twitter mention graph
  - Also want to show top hashtag in each component
- One input stream of incoming tweets – 32k per second
- One query stream of usernames – 1 every 100ms
Naiad is fast for iterative graph computation

Thoughts, Comments, and Further Questions

• Is this kind of real-time querying useful?
  • Many analytics workloads can be performed in batch mode
Thoughts, Comments, and Further Questions

• Is this kind of real-time querying useful?
  • Many analytics workloads can be performed in batch mode

• Extremely sensitive to ‘micro-stragglers’ – single worker stalls
  • Requires significant system optimization and tuning, not an easy ‘run anywhere’ system
Thoughts, Comments, and Further Questions

• Is this kind of real-time querying useful?
  • Many analytics workloads can be performed in batch mode

• Extremely sensitive to ‘micro-stragglers’ – single worker stalls
  • Requires significant system optimization and tuning, not an easy ‘run anywhere’ system

• Requires good partitions for good parallelism – relies on developer
Thoughts, Comments, and Further Questions

• Is this kind of real-time querying useful?
  • Many analytics workloads can be performed in batch mode

• Extremely sensitive to ‘micro-stragglers’ – single worker stalls
  • Requires significant system optimization and tuning, not an easy ‘run anywhere’ system

• Requires good partitions for good parallelism – relies on developer

• Cannot support dynamic computation graphs
Thoughts, Comments, and Further Questions

• Timely Dataflow allows for Differential Dataflow – rather than recompute on all the data, just recompute on the part that changed
Thoughts, Comments, and Further Questions

• Timely Dataflow allows for **Differential Dataflow** – rather than recompute on all the data, just recompute on the part that changed

• Very well suited to real-time graph computation where number of iterations might be dynamic (e.g. BFS)
Thoughts, Comments, and Further Questions

• Timely Dataflow allows for Differential Dataflow – rather than recompute on all the data, just recompute on the part that changed

• Very well suited to real-time graph computation where number of iterations might be dynamic (e.g. BFS)

• Matches performance of many specialized systems of the time
Questions?