Verified Time-Aware Stream Processing

Rafael Castro G. Silva

rafaelcgs10@gmail.com

Department of Computer Science
University of Copenhagen

02/11/2023

What is this PhD about?

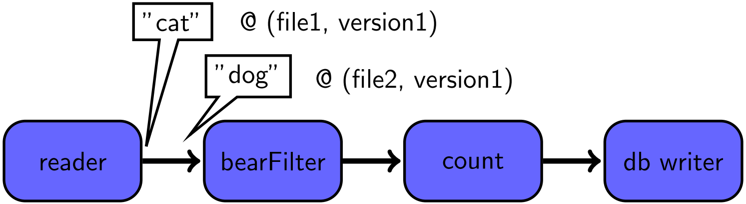
- Distributed Systems
  - Stream processing frameworks
    - Dataflow models
      - Timely Dataflow
- Formal Methods
  - Verification using proof assistants
    - Isabelle proofs
      - Verified and executable code

Contents

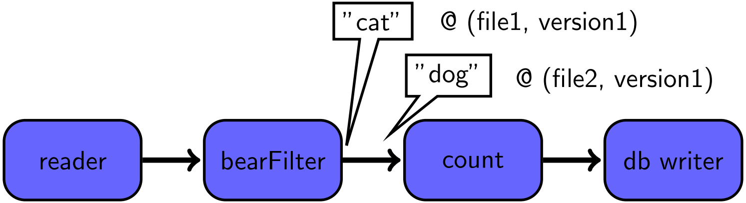
- Timely Dataflow
- Progress Tracking
- Verified Progress Tracking
- Verified Timely Dataflow
- Questions, comments and suggestions

Timely Dataflow

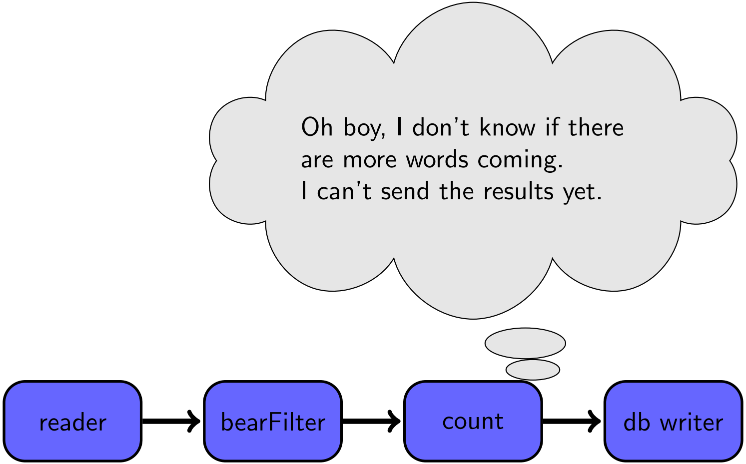
Example of a conventional Dataflow



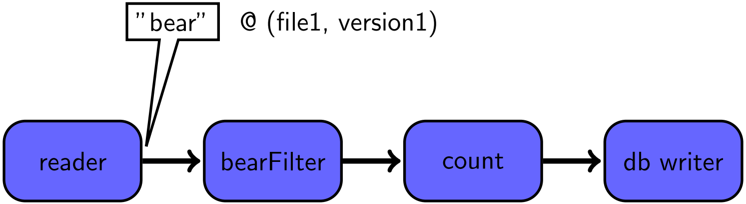
Example of a conventional Dataflow



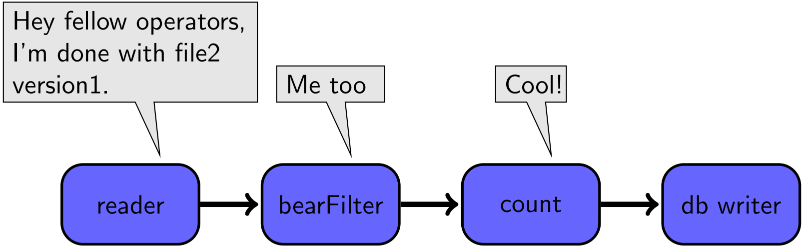
Example of a conventional Dataflow



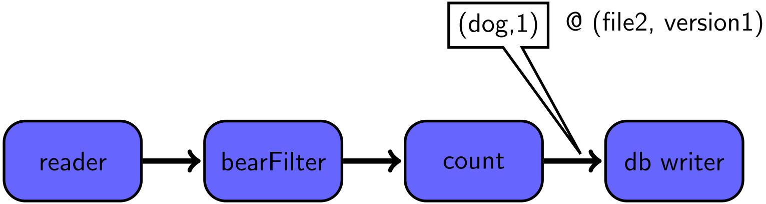
Example of a conventional Dataflow



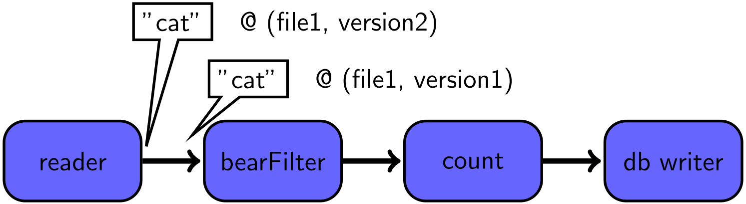
Example of a conventional Dataflow



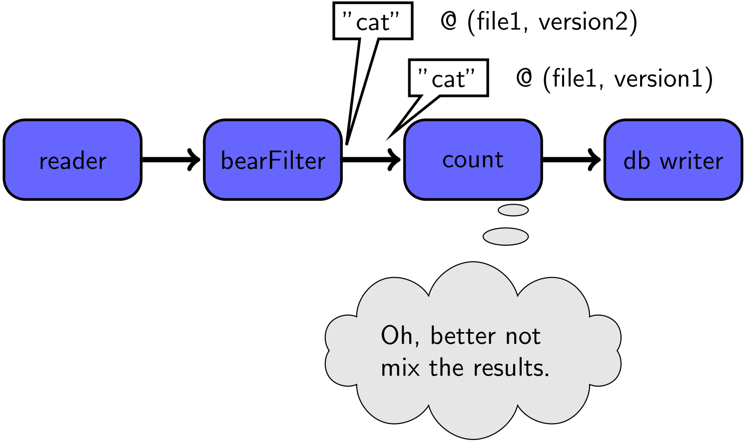
Example of a conventional Dataflow



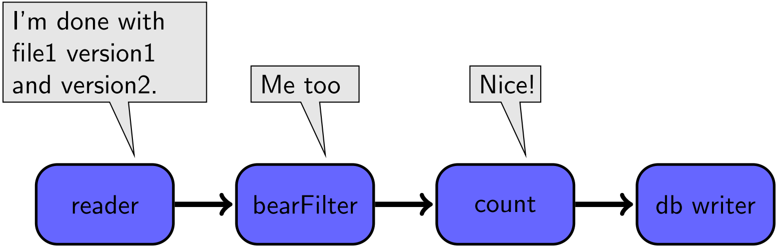
Example of a conventional Dataflow



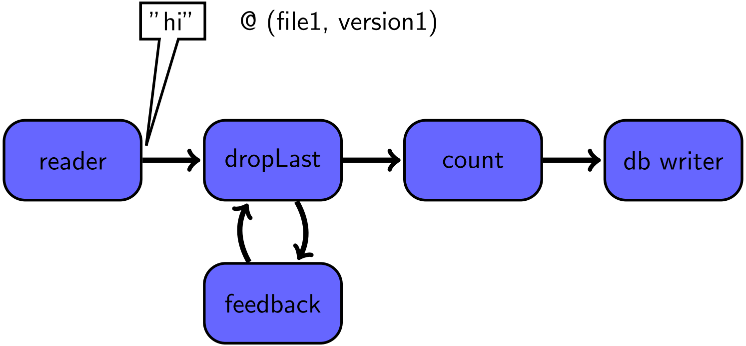
Example of a conventional Dataflow



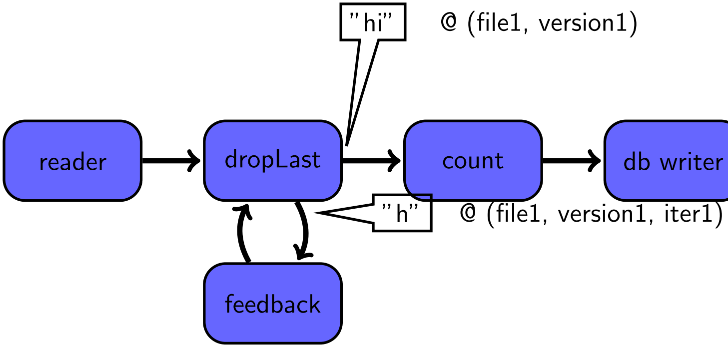
Example of a conventional Dataflow



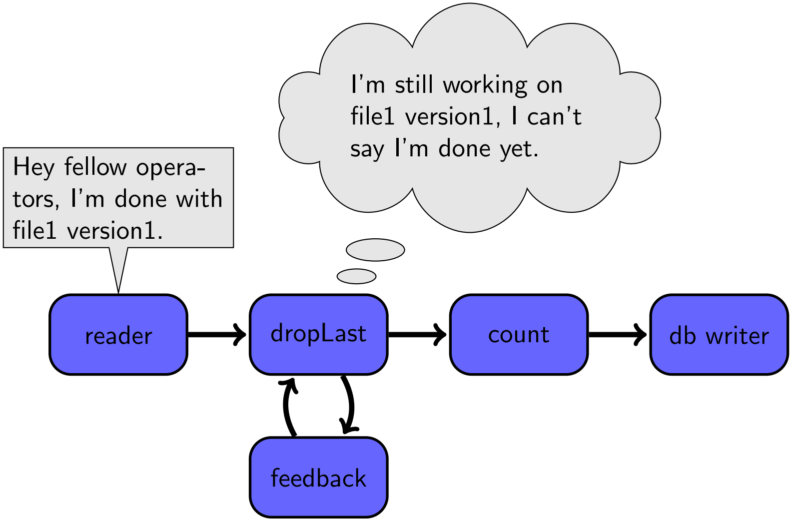
Example of a conventional Dataflow with a cycle



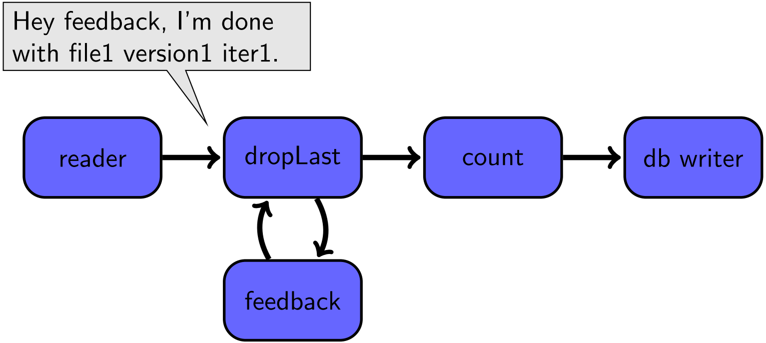
Example of a conventional Dataflow with a cycle



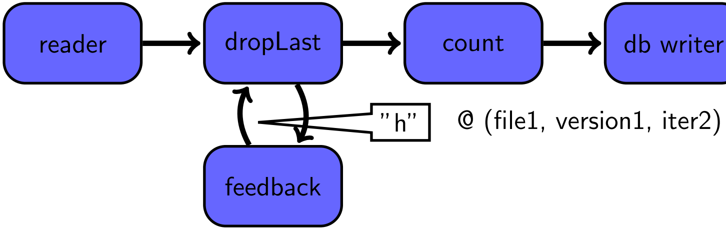
Example of a conventional Dataflow with a cycle



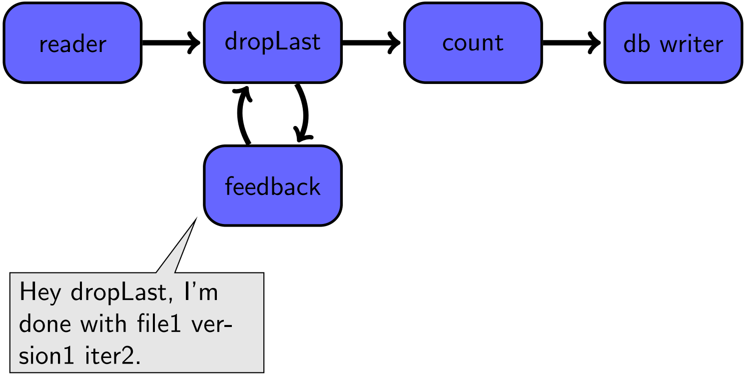
Example of a conventional Dataflow with a cycle



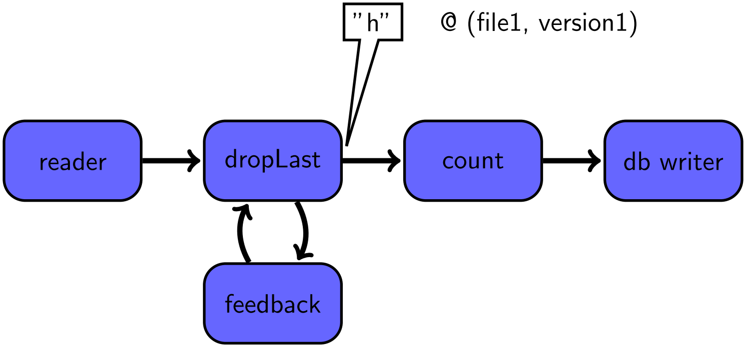
Example of a conventional Dataflow with a cycle



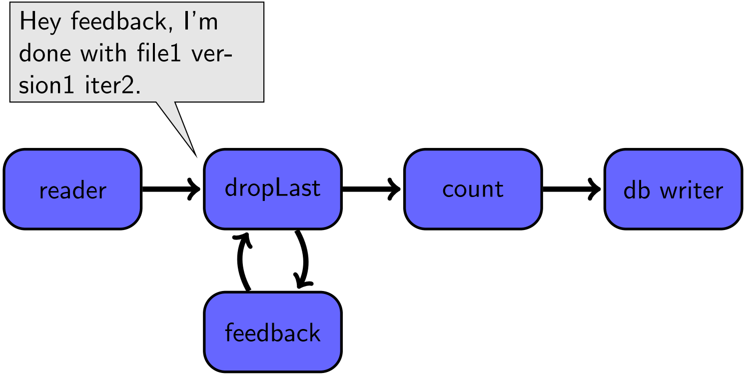
Example of a conventional Dataflow with a cycle



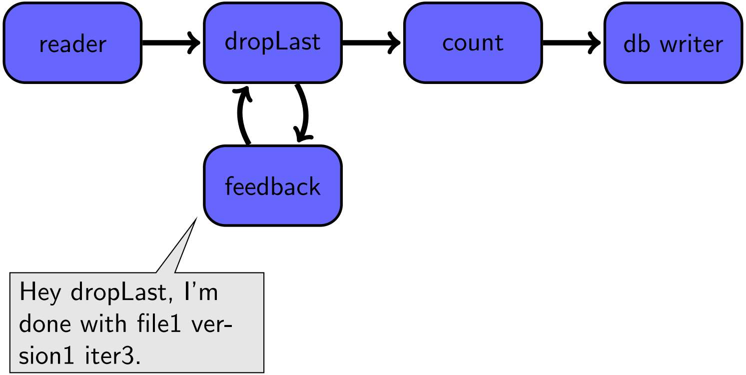
Example of a conventional Dataflow with a cycle



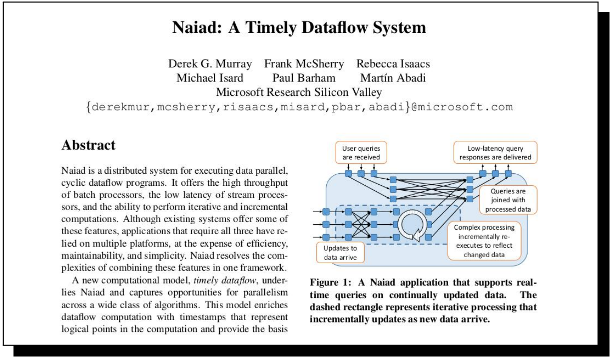
Example of a conventional Dataflow with a cycle



Example of a conventional Dataflow with a cycle



Timely Dataflow


Naiad: A Timely Dataflow System

Derek G. Murray Frank McSherry Rebecca Isaacs
Michael Isard Paul Barham Martín Abadi
Microsoft Research Silicon Valley
{derekmur,mcsherry,risaacs,misard,pbar,abadi}@microsoft.com

Abstract

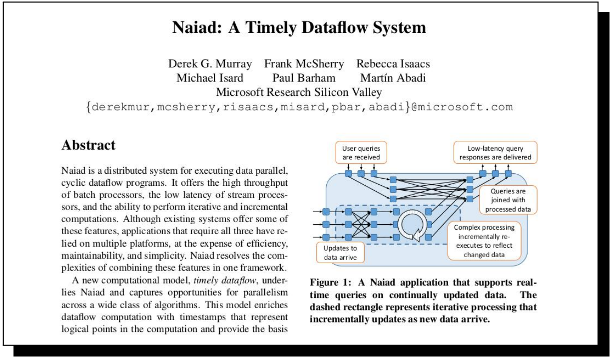
Naiad is a distributed system for executing data parallel, cyclic dataflow programs. It offers the high throughput of batch processors, the low latency of stream processors, and the ability to perform iterative and incremental computations. Although existing systems offer some of these features, applications 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

Figure 1: A Naiad application that supports real-time queries on continually updated data. The dashed rectangle represents iterative processing that incrementally updates as new data arrive.

- Iterative computation (cycles)
- Reports on the presence of data instead of the absence
- Two implementations: C# (Naiad) and Rust

Timely Dataflow



Naiad: A Timely Dataflow System

Derek G. Murray Frank McSherry Rebecca Isaacs
Michael Isard Paul Barham Martín Abadi
Microsoft Research Silicon Valley
{derekmur,mcsherry,risaacs,misard,pbar,abadi}@microsoft.com

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

Figure 1: A Naiad application that supports real-time queries on continually updated data. The dashed rectangle represents iterative processing that incrementally updates as new data arrive.

- Iterative computation (cycles)
- Reports on the presence of data instead of the absence
- Two implementations: C# (Naiad) and Rust

Timely Dataflow

Naiad: A Timely Dataflow System

Derek G. Murray   Frank McSherry   Rebecca Isaacs
Michael Isard   Paul Barham   Martín Abadi
Microsoft Research Silicon Valley
{derekmur,mcsherry,risaacs,misard,pbar,abadi}@microsoft.com

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

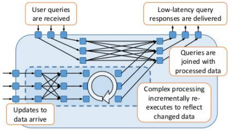


Figure 1: A Naiad application that supports real-time queries on continually updated data. The dashed rectangle represents iterative processing that incrementally updates as new data arrive.

- Iterative computation (cycles)
- Reports on the presence of data instead of the absence
- Two implementations: C# (Naiad) and Rust

Timely Dataflow

Naiad: A Timely Dataflow System

Derek G. Murray Frank McSherry Rebecca Isaacs
Michael Isard Paul Barham Martín Abadi
Microsoft Research Silicon Valley
{derekmur,mcsherry,risaacs,misard,pbar,abadi}@microsoft.com

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

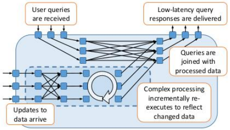


Figure 1: A Naiad application that supports real-time queries on continually updated data. The dashed rectangle represents iterative processing that incrementally updates as new data arrive.

- Iterative computation (cycles)
- Reports on the presence of data instead of the absence
- Two implementations: C# (Naiad) and Rust

Progress Tracking

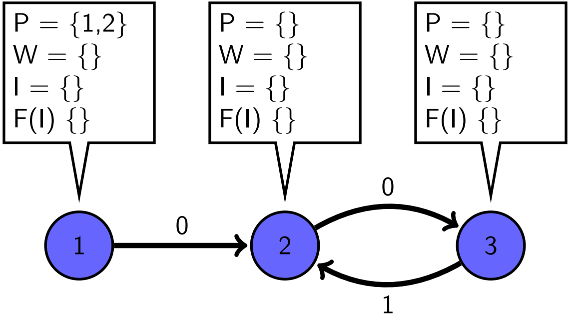
Progress Tracking Protocollllll

- Two core components: Local Propagation and Distributed Exchange
  - Local: localy compute the frontiers
    - *A frontier is a lower bound on the timestamps that may appear at the operator instance inputs*
  - Exchange the timestamps between workers
- For every worker and for every location of the graph a conservative approximation of which timestamps may still arrive (a.ka. frontier)

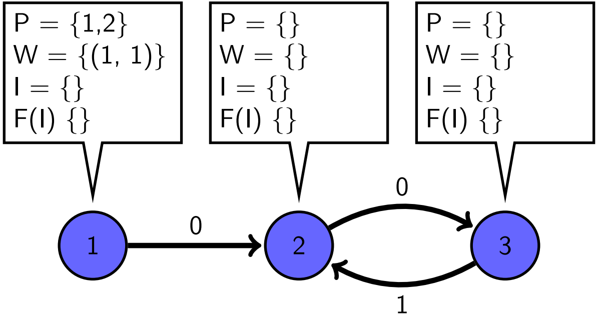
Progress Tracking ProtocolIIII

- Two core components: Local Propagation and Distributed Exchange
  - Local: localy compute the frontiers
    - *A frontier is a lower bound on the timestamps that may appear at the operator instance inputs*
  - Exchange the timestamps between workers
- For every worker and for every location of the graph a conservative approximation of which timestamps may still arrive (a.ka. frontier)

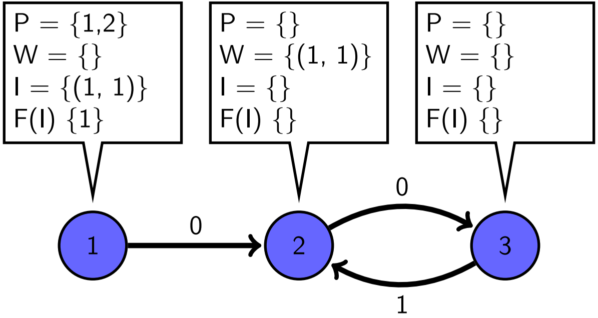
Local Propagation Example



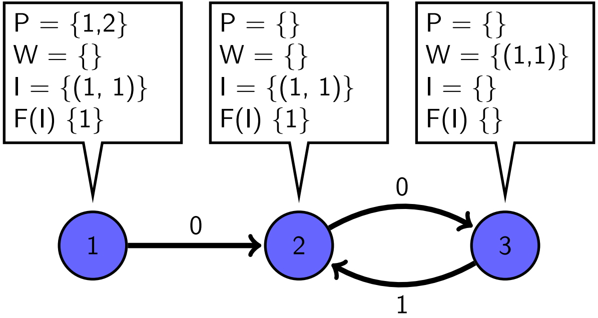
Local Propagation Example



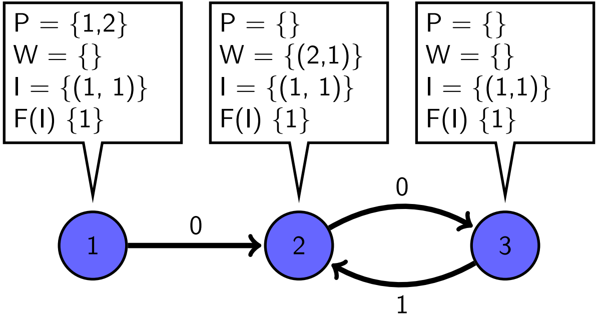
Local Propagation Example



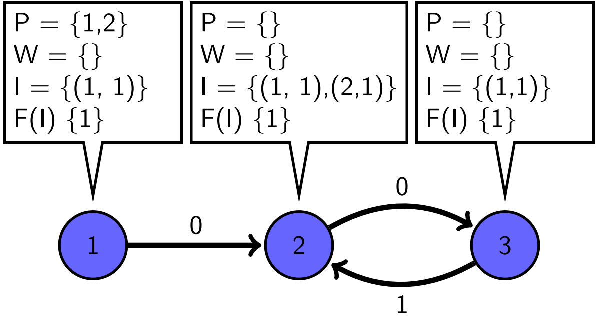
Local Propagation Example



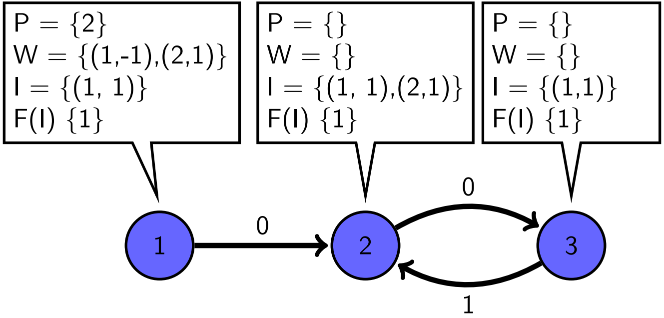
Local Propagation Example



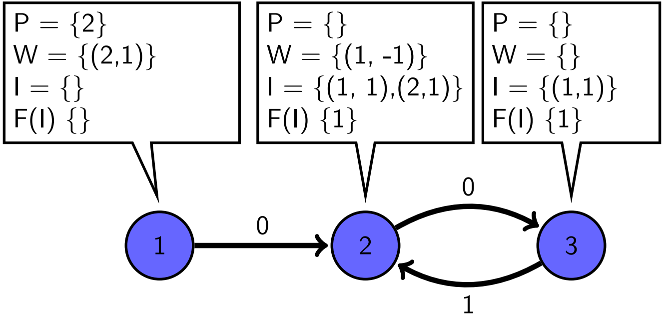
Local Propagation Example



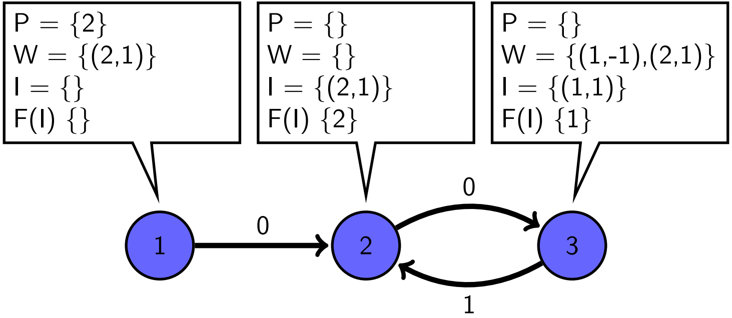
Local Propagation Example



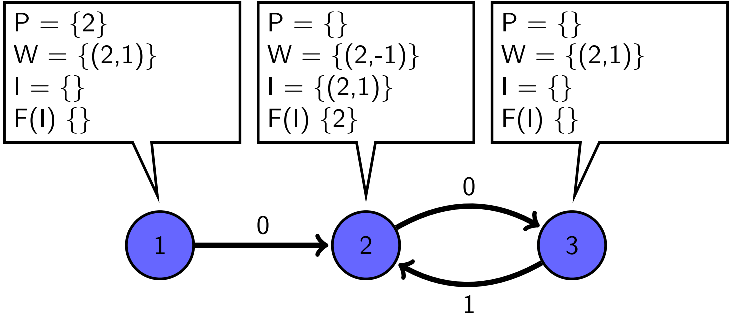
Local Propagation Example



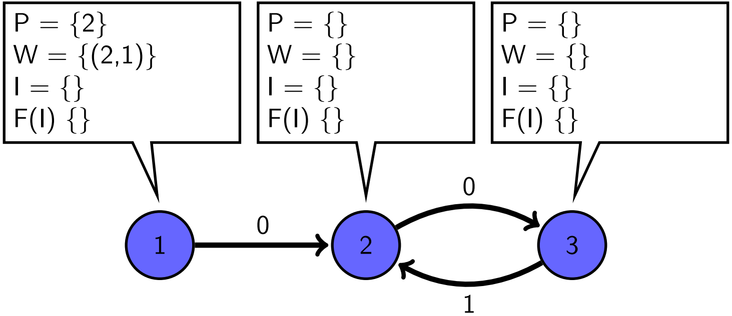
Local Propagation Example



Local Propagation Example



Local Propagation Example



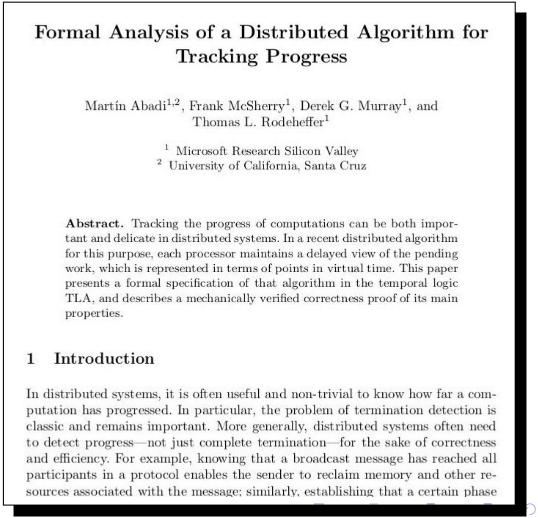
Verified Progress Tracking

Abadi et al.'s exchange protocol

- Exchange Protocol Verification:
  - Presented by Abadi et al. in TLA+

Distribute Safety

If a timestamp is vacant in one worker (now), then that timestamp and any lesser it is vacant for the global system state (now and forever)



Formal Analysis of a Distributed Algorithm for Tracking Progress

Martín Abadi[1,2], Frank McSherry[1], Derek G. Murray[1], and Thomas L. Rodeheffer[1]

[1] Microsoft Research Silicon Valley
[2] University of California, Santa Cruz

Abstract. Tracking the progress of computations can be both important and delicate in distributed systems. In a recent distributed algorithm for this purpose, each processor maintains a delayed view of the pending work, which is represented in terms of points in virtual time. This paper presents a formal specification of that algorithm in the temporal logic TLA, and describes a mechanically verified correctness proof of its main properties.

1 Introduction

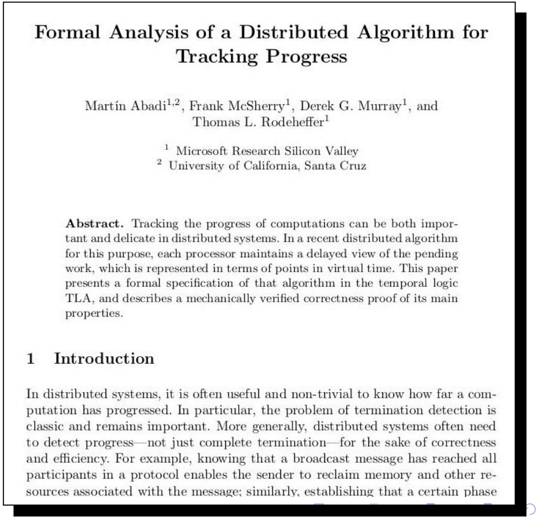
In distributed systems, it is often useful and non-trivial to know how far a computation has progressed. In particular, the problem of termination detection is classic and remains important. More generally, distributed systems often need to detect progress—not just complete termination—for the sake of correctness and efficiency. For example, knowing that a broadcast message has reached all participants in a protocol enables the sender to reclaim memory and other resources associated with the message; similarly, establishing that a certain phase

Abadi et al.'s exchange protocol

- Exchange Protocol Verification:
  - Presented by Abadi et al. in TLA+

Distribute Safety

If a timestamp is vacant in one worker (now), then that timestamp and any lesser it is vacant for the global system state (now and forever)



Formal Analysis of a Distributed Algorithm for Tracking Progress

Martín Abadi[1,2], Frank McSherry[1], Derek G. Murray[1], and Thomas L. Rodeheffer[1]

[1] Microsoft Research Silicon Valley
[2] University of California, Santa Cruz

Abstract. Tracking the progress of computations can be both important and delicate in distributed systems. In a recent distributed algorithm for this purpose, each processor maintains a delayed view of the pending work, which is represented in terms of points in virtual time. This paper presents a formal specification of that algorithm in the temporal logic TLA, and describes a mechanically verified correctness proof of its main properties.

1 Introduction

In distributed systems, it is often useful and non-trivial to know how far a computation has progressed. In particular, the problem of termination detection is classic and remains important. More generally, distributed systems often need to detect progress—not just complete termination—for the sake of correctness and efficiency. For example, knowing that a broadcast message has reached all participants in a protocol enables the sender to reclaim memory and other resources associated with the message; similarly, establishing that a certain phase

Verified Progress Tracking for Timely Dataflow

- Exchange Protocol Verification:
  - Refined version
  - Same safety property
- Local Propagation Verification

Local Safety

If all worklists neither contain timestamp t nor smaller timestamps, then all locations know whether they may encounter t in the future.

Verified Progress Tracking for Timely Dataflow

Matthias Brun ✉
Department of Computer Science, ETH Zürich, Switzerland

Sára Decova
Department of Computer Science, ETH Zürich, Switzerland

Andrea Lattuada ✉
Department of Computer Science, ETH Zürich, Switzerland

Dmitriy Traytel ✉
Department of Computer Science, University of Copenhagen, Denmark

Abstract

Large-scale stream processing systems often follow the dataflow paradigm, which enforces a program structure that exposes a high degree of parallelism. The Timely Dataflow distributed system supports expressive cyclic dataflows for which it offers low-latency data- and pipeline-parallel stream processing. To achieve high expressiveness and performance, Timely Dataflow uses an intricate distributed protocol for tracking the computation's progress. We modeled the progress tracking protocol as a combination of two independent transition systems in the Isabelle/HOL proof assistant. We specified and verified the safety of the two components and of the combined protocol. To this end, we identified abstract assumptions on dataflow programs that are sufficient for safety and were not previously formalized.

2012 ACM Subject Classification Security and privacy → Logic and verification; Computing methodologies → Distributed algorithms; Software and its engineering → Data flow languages

Keywords and phrases safety, distributed systems, timely dataflow, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.ITP.2021.10

Related Version Extended report that provides proof sketches and further formalization details.
Full Version: https://www.github.com/matthias-brun/progress-tracking-formalization/raw/main/rep.pdf [9]

Supplementary Material *Software (Isabelle Formalization)*: https://www.isa-afp.org/entries/Progress_Tracking.html

Verified Progress Tracking for Timely Dataflow

- Exchange Protocol Verification:
  - Refined version
  - Same safety property
- Local Propagation Verification

Local Safety

If all worklists neither contain timestamp t nor smaller timestamps, then all locations know whether they may encounter t in the future.

Verified Progress Tracking for Timely Dataflow

Matthias Brun ✉
Department of Computer Science, ETH Zürich, Switzerland

Sára Decova
Department of Computer Science, ETH Zürich, Switzerland

Andrea Lattuada ✉
Department of Computer Science, ETH Zürich, Switzerland

Dmitriy Traytel ✉
Department of Computer Science, University of Copenhagen, Denmark

Abstract

Large-scale stream processing systems often follow the dataflow paradigm, which enforces a program structure that exposes a high degree of parallelism. The Timely Dataflow distributed system supports expressive cyclic dataflows for which it offers low-latency data- and pipeline-parallel stream processing. To achieve high expressiveness and performance, Timely Dataflow uses an intricate distributed protocol for tracking the computation's progress. We modeled the progress tracking protocol as a combination of two independent transition systems in the Isabelle/HOL proof assistant. We specified and verified the safety of the two components and of the combined protocol. To this end, we identified abstract assumptions on dataflow programs that are sufficient for safety and were not previously formalized.

2012 ACM Subject Classification Security and privacy → Logic and verification; Computing methodologies → Distributed algorithms; Software and its engineering → Data flow languages

Keywords and phrases safety, distributed systems, timely dataflow, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.ITP.2021.10

Related Version Extended report that provides proof sketches and further formalization details.
Full Version: `https://www.github.com/matthias-brun/progress-tracking-formalization/raw/main/rep.pdf` [9]

Supplementary Material *Software (Isabelle Formalization)*: `https://www.isa-afp.org/entries/Progress_Tracking.html`

Verified Timely Dataflow

Verified Timely Dataflow

General goal

Contribute to an ongoing formalization effort using the Isabelle proof assistant towards the first formally verified system for the analysis of big data.

Specific goals

1. Termination of the propagation protocol
2. Executable Progress Tracking Protocol
3. Data plane
4. Scopes
5. DSL
6. Experiments

Verified Timely Dataflow

General goal

Contribute to an ongoing formalization effort using the Isabelle proof assistant towards the first formally verified system for the analysis of big data.

Specific goals

1. Termination of the propagation protocol
2. Executable Progress Tracking Protocol
3. Data plane
4. Scopes
5. DSL
6. Experiments

Termination of the propagation protocol

- *propagate* is a relation between two system configurations and also parameterized by location and a timestamp:
  - Propagating the timestamp *t* in location *loc* the configuration *c1* turns into *c2*
- We aim to show that it's impossible cannot propagate forever
- We've already managed to show that for a fixed timestamp it cannot be propagate forever

Executable Progress Tracking Protocol

- The propagation protocol is already executable (Sara Decova's master thesis)
- The Exchange protocol also should be made executable
  - Prove that the functions are consistent with the relational counterparties

Data plane

- Nodes must be real operators: they execute user defined functions over the data
- Edges must be channels: they are streams with push and pull API
  - Must exchange data between different workers
- Low level operator constructors (define operators for any amount of inputs and outputs)
- A set of generally useful built-in operators: filter, filter partition, count, aggregate, etc
- The Safety properties again!
  - Use safety to show properties of operators and dataflows
    - Ex: show that *count* doesn't mix the results of different (comparable) timestamps
- Executable from the start

Scopes

- Hierarchal dataflows: an operator can also be a sub-dataflow inside of a dataflow
- This is designed as scopes
  - An additional timestamp dimension
- Another propagation (upwards and downwards)
- The Safety properties once again!

DSL

- Writting dataflow programs in Isabelle is quite inconvenient
- A suitable domain-specific language (DSL)

Experiments

1. How faithful is the verified model to the Rust implementation?
2. Are there any bugs in the Rust implementation not present in the verified code?
3. How does the performance of the verified code compares to the Rust implementation?

Questions, comments and suggestions