Differential durability: fault-tolerant distributed differential computation

†Andrea Lattuada, *Vasiliki Kalavri, †Moritz Hoffmann
Systems Group, Department of Computer Science, ETH Zürich
firstname.lastname@inf.ethz.ch, †*PhD student, *poster presenter*

Frank McSherry
Unaffiliated
mcsherry@gmail.com

Introduction Fault-tolerance mechanisms for streaming computations can be expensive due to the requirement of capturing a consistent state of the system, including in-flight messages and operator state. We propose an efficient fault-tolerance and durability mechanism for an existing incremental computational model for distributed computation, *Differential Dataflow*[DD]. We leverage the intrinsic properties of the model to achieve asynchronous replication and fault-tolerance, with reduced overhead and recovery cost.

Motivation Operator state in streaming jobs is very valuable and should be guarded against failure: lack of fault-tolerance would result in incorrect results after recovery. Additionally, streaming jobs run for long periods of time, accumulating state over several days or even months: reprocessing all input in the case of failures would be prohibitively expensive and time-consuming. The common approaches to fault-tolerance based on snapshots or active replication can have a significant negative impact on latency and overall performance. We explore a technique that affords reduced impact on performance by moving the replication out of the critical path, thanks to the properties of the chosen computational model.

Differential Dataflow Differential Dataflow represents inputs and outputs as collections of items that evolve with (logical) time. Each collection in differential dataflow reflects an append-only log of immutable updates indexed by the logical timestamp. Updates are committed atomically every time computation for a certain (logical) time is completed. For this reason, each collection has known consistent states after each commit: these are natural rollback points in case of failure. Additionally, (i) the state for an operator corresponds to its input collection, indexed for efficient access, and (ii) operator output is deterministic given a certain input. Thanks to these properties, incomplete computation output for a logical timestamps can be dropped and reconstructed from the operator input.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroSys'17,
Copyright © ACM [to be supplied]. . . $15.00.
http://dx.doi.org/10.1145/.

Our approach

Replication The immutable and append-only nature of the collection log enables asynchronus replication of the state with minimal coordination. A consensus-based commit process is not required: as soon as a batch of updates for a certain timestamp has been replicated to enough machines to ensure resilience to failure and network partitions, we can promote that batch to a safe rollback point. Because replication is asynchronous, this approach side-steps the need to select a tradeoff between snapshot volume and time needed to recover from a failure.

Recovery The recovery process starts new copies of the failed operators in the surviving nodes where their input collections have been replicated. The logs are rolled back to a safe point corresponding to a complete (logical) timestamp, and execution is resumed.

Log garbage collection Accumulated deltas can take a large amount of space, requiring periodic clean-up. Garbage collection and log compaction can be performed by collapsing deltas that fall behind a global safe rollback point. This operation is the most expensive, but can be performed lazily (with a tradeoff with memory consumption) and off the critical path.

Other features The ability to selectively roll-back operators to any previous consistent state enables a variety of applications beyond fault-tolerance. A section of the computation can be rewinded and then resumed with a tweaked version of the business logic to compare the outcomes. In a similar way, the replicated logs can serve as the basis for speculative execution of parallel tasks to mitigate the adverse effects of stragglers.

References

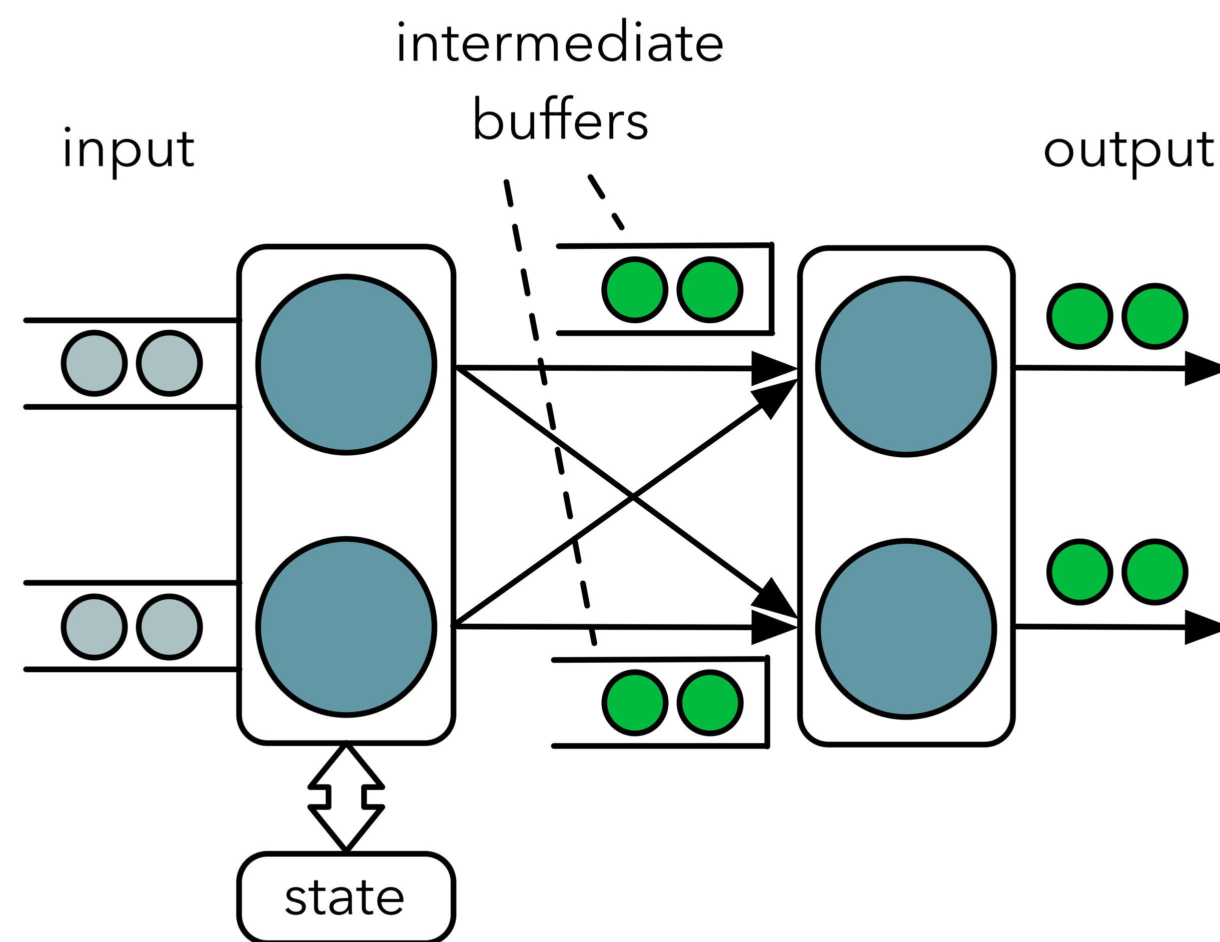
[DD] Frank Mcsherry, Derek G. Murray, Rebecca Isaacs, Michael Isard, *Differential dataflow*

Differential Durability: fault-tolerant distributed differential computation

†Andrea Lattuada, †Vasiliki Kalavri, †Moritz Hoffmann, ‡Frank McSherry

MOTIVATION

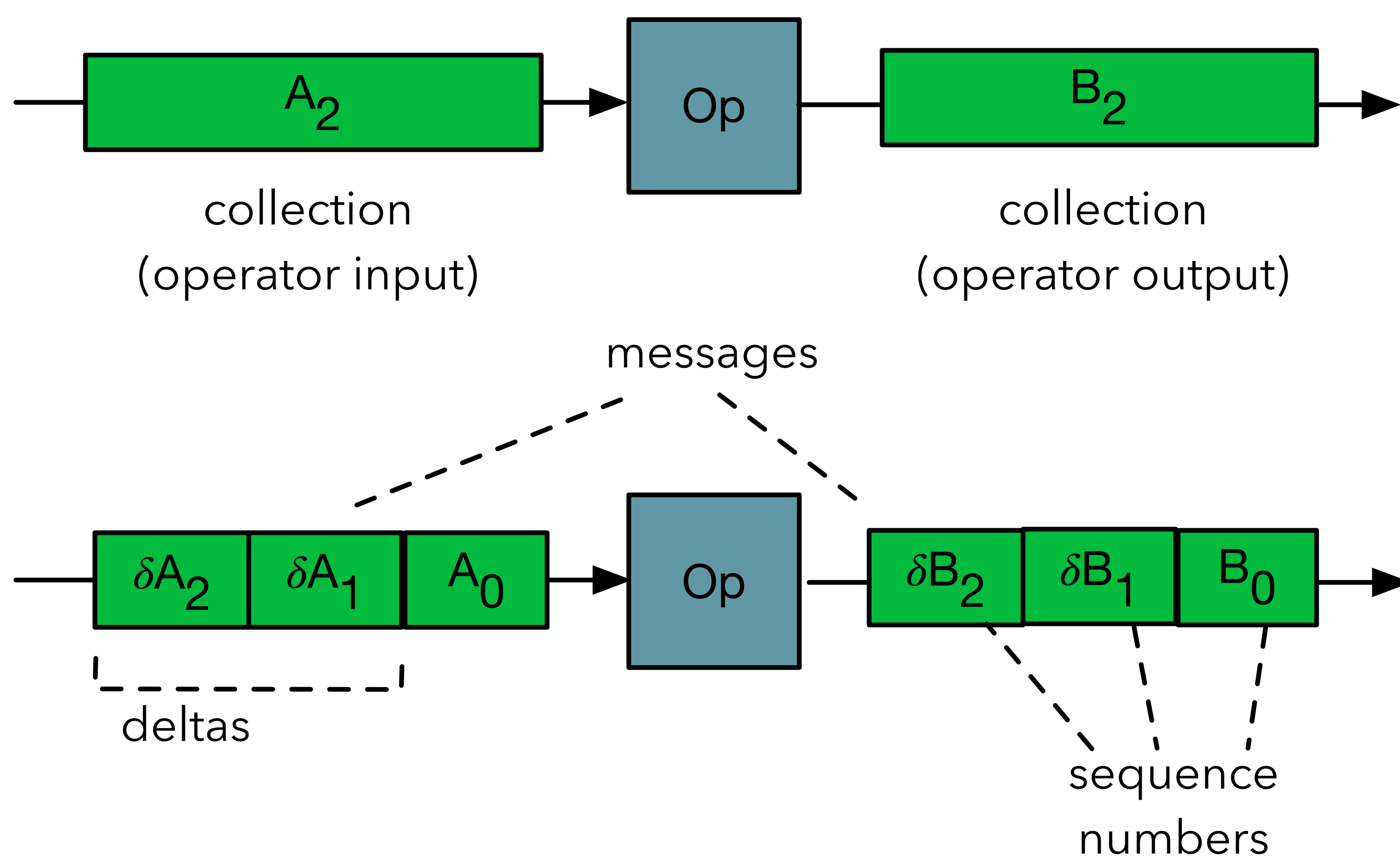
- Any non-trivial **streaming** application needs to maintain **state**
  - rolling aggregation, windows
- Streaming applications run for days, months, even years
- Distributed systems *will* fail
  - how can we guard state against failures and guarantee correct results after recovery?



EXISTING FAULT-TOLERANCE MECHANISMS

- Snapshot-based
  - requires (non-blocking) synchronisation
- Active replication to identical standbys
  - requires 2x resources
- Atomic Transactions for message logs state change
  - requires expensive synchronisation

DIFFERENTIAL DATA PROCESSING MODEL

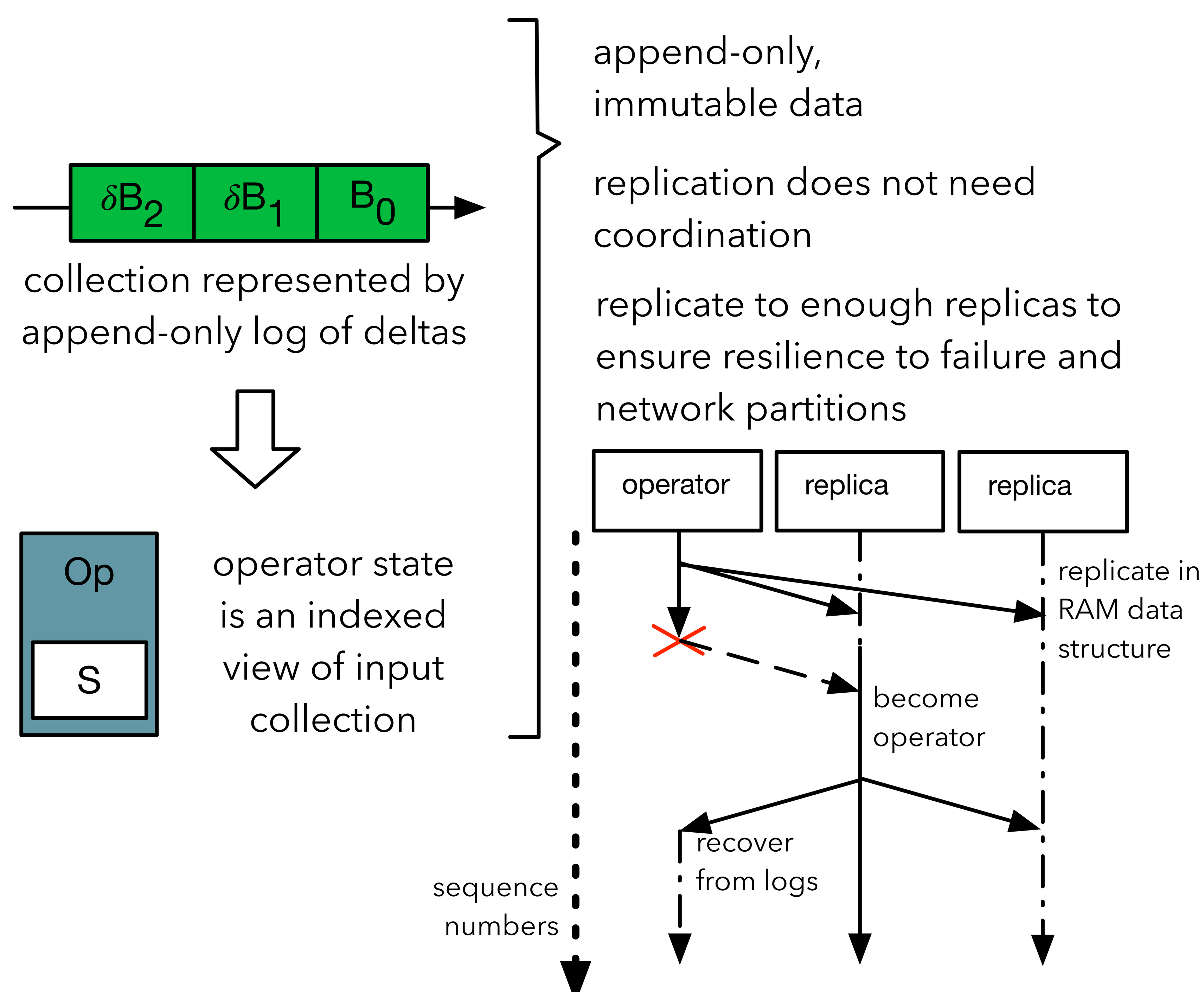


Differential Dataflow
McSherry, et al. [CIDR 2013]

Operators are **functional**, deterministically transform input collection to output collection (represented as sequence of deltas).

- Data-parallel, incremental processing of large-scale data
- Support for iterative and interactive queries
- Fine-grained synchronisation mechanism among workers
- Millisecond-scale latency

OUR APPROACH: ASYNCHRONOUS REPLICATION



RECOVERY

- functional computation, can always recover from input (via rollback)
- replicated data always consistent (append only)
- re-computation can be avoided if results are available in log

LOG COMPACTION

- compact state to safe sequence no.
- compact collections to safe sequence no.

asynchronous; performed by replicas independently, *without synchronisation*

FUTURE WORK

- time travel
- dynamic rescaling

†Systems Group, Department of Computer Science, ETH Zurich
firstname.lastname@inf.ethz.ch

‡Unaffiliated
mcsherry@gmail.com