Minimizing Latency in Fault-Tolerant Distributed Stream Processing Systems

Andrey Brito¹, Christof Fetzer¹, Pascal Felber²

¹ Technische Universität Dresden, Germany
² Université de Neuchâtel, Switzerland

ICDCS'09, June 23rd, 2009
Goal

Minimize the cost of logging/checkpointing in event stream processing systems

**Contribution:** Usage of an speculation framework based on transactional memory to overlap logging and processing
Motivation (1)

• Event stream applications
  – Directed acyclic graph of operators
  – Some operators don't keep state
    • Trivially parallelizable
  – Some do keep state
    • Not trivially parallelizable
  – Sometimes they are order sensitive
    • Need to process events sequentially, maybe even waiting for the order to be restored
Application example
Application example

Events based on non-deterministic decision

Events are out!

ICDCS'09, 23.06.09  Minimizing latency in fault-tolerant DSMS  Slide 5 of 55
Application example
Application example

Restore checkpoint.
Application example

Ask upstream node to replay missing ones.
Application example

Processing some events again.

Publisher A

Filter n

A6

Publisher B

Filter n

B8

Processor1

B3 B2 A4 B1

Processor2

Output Adapter
Application example

Events reflect different decisions.

Incomplete log of non-deterministic decisions → no repeatability
Motivation (2)

- Fault-tolerant event stream applications
  - Precise recovery
  - Even if order does not matter, **repeatability** does
  - Non-determinism
    - Input order from different streams
    - Non-determinism in processing (multi-threading, time, random numbers)
  - Log or checkpoint before each output
Logging is expensive

![Bar chart showing total latency in microseconds for different configurations of disks (1 disk, 2 disks, 3 disks) with non-speculative logging.]
My solution

- Speculate...
- ... to parallelize stateful components
- ... to not have to wait for events
- ... to not have to wait for logging
Outline

• How the speculation works
• Logging algorithm
• Experiments
• Final remarks
How the speculation works

- **Base: TinySTM**
  - Some extra features added
  - But same basic rule: “it appears to be atomic”

- **Goal: track accesses to shared memory**
  - Instrumentation
    - Reads and writes are intercepted
    - Hold back writes, validate reads until all dependencies satisfied
Speculative execution: parallelization
Speculative execution: parallelization
Speculative execution: parallelization
Speculative execution: parallelization
Speculative execution: parallelization
Speculative execution: parallelization
Logging algorithm

- Operator enqueues all events & decisions

- N+1 threads for N disks
  - One groups requests in a buffers
  - The others write their buffers to disk
Logging algorithm
Logging algorithm
Logging algorithm
Logging algorithm
Logging algorithm
Logging algorithm

E is here waiting.
Logging algorithm
Logging algorithm
Logging algorithm

Operator

update(E)
Logging algorithm
Logging algorithm
Logging algorithm

Events based on non-deterministic decision

Events are out!
Logging algorithm
Logging algorithm
Logging algorithm

Diagram: Flowchart showing a process involving filters (Filter 1 and Filter n) directing traffic to a processor (Processor 1), which then feeds into a checkpoint/logging system. The diagram illustrates the flow of data through these components.
Logging algorithm

Diagram:
- Filter 1
- Filter n
- Processor 1
- STATE
- Checkpoint/Logging

Flow:
1. Data from Filters 1 to n to Processor 1
2. Data from Processor 1 to Checkpoint/Logging
Logging algorithm

Diagram:
- Filter 1
- Filter n
- Processor 1
- STATE
- Checkpoint/Logging
- Arrows: 1 ➔ 2 ➔ 3
Logging algorithm
Logging algorithm

1. Filter 1
2. Filter n
3. Processor 1
4. Checkpoint/Logging
5. STATE
Logging algorithm

Diagram:
- Filter 1
- Filter n
- Processor 1
- STATE
- Checkpoint/Logging

Arrows:
1. Filter 1 to Processor 1
2. Filter n to Processor 1
3. Downward arrow from Processor 1 to Checkpoint/Logging
4. Upward arrow from Processor 1 to Checkpoint/Logging
5. Arrow from Filter 1 to Filter n
6. Arrow from Processor 1 to Checkpoint/Logging
Logging algorithm
Speculative processing + Logging

• From the original node's viewpoint
  – Emit outputs as speculative
  – When logging requests are acknowledged, emit final

• The next downstream node
  – If speculative event modifies some state, keep track
    • Outputs that consider that part of the state are speculative
    • Speculative status is contagious
Speculation + Logging
Experiments

- Parallelization: benefits & STM's overheads
- Optimism control
- Overlapping processing and logging
Speculation costs & speed-ups

- Non-speculative
- 4 threads
- 8 threads

Hardware: Sun T1000

Speculation creation-commit-disposal overheads.

Few shared-memory accesses.

Amdahl's law influence.
Controlling optimism
Controlling optimism

![Diagram showing processors and NEXT values](image)
Controlling optimism

Hardware: SUN T1000

State size varies between 1 and 20.

Size=1: concurrent executions will conflict.

Size=20: considerable parallelism.
Motivation for distributed speculation: logging costs

- **Non-speculative**: only stable events are sent.
- **Speculative**: send events before logging is finished.

### Graph

- **Y-axis**: Total latency (microseconds)
- **X-axis**: Configuration
  - 1 disk
  - 2 disks
  - 3 disks
  - Sim 10000
  - Sim 5000

- **Legend**:
  - Non-speculative
  - Speculative

2 components do logging.

Non-speculative: only stable events are sent.

Speculative: send events before logging is finished.
Accumulated gains

X axis: number of components logging.

Even in a SAN/WAN, the shapes would look similar.

For deterministic components: add “fixed” latency.
Final remarks - Parallelization

• Parallelization through speculation
  – Easier, less bugs
    • Programmer does not need to fight with locks
    • Keeps sequential semantics
  – Waste of resources reduced with optimism control

• Overhead can be much lower with hardware support for TM (e.g., ASF)
Final remarks - Logging

• Overlap logging with processing
  – Independent of available parallelism
  – Distributed speculation possible due to less aborts
  – But do not let speculative results get out of the system

• In combination with speculative parallelization may even reduce logging
Thank you!

http://streammine.inf.tu-dresden.de
http://wwwse.inf.tu-dresden.de