Speculative Concurrent Processing with Transactional Memory in the Actor Model

Yaroslav Hayduk, Anita Sobe, Derin Harmanci, Patrick Marlier and Pascal Felber

University of Neuchatel, Switzerland

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 318693
A bit of background: The Actor Model

- Hewitt & Baker (IFIP Congress’77) – „Laws for Communicating Parallel Processes“

Motivated by the prospect of highly parallel computing machines with many microprocessors + own local memory
OOP and actors: Communication

- Everything is an actor (VS an object)
- Asynchronous message passing
- Has access to its local state only
- Strong encapsulation
- Inherently concurrent
OOP and actors: Communication

Object A

ObjectB.publicMethod()  
ObjectB.publicField=10

Object B

direct access

Illegal: strong encapsulation

Object A

ActorB.publicField = 10

ActorA [SendMessageTo] ActorB

asynchronous message passing

Object A
Problem statement

- Sequential processing of messages limits performance & throughput
- Multiple actors participating in the same coordinated transaction block, causing message processing delays
Main contributions

- A method for processing many messages concurrently using STM, and
- A method for removing excessive blocking, associated with the processing of coordinated transactions

⇒ both methods preserve the semantics of the Actor Model
A current solution

- Habanero Scala - Shams et al. (Scala Days’12) – Habanero Scala
  ⇒ async-finish programming model
- Main strength: processes parts of one message concurrently
Case 1: Concurrent message processing

B
Contains 2

Remove 9

Actor C ("List" Actor)
Local Actor state
2 → 5 → 6 → 8 → 9
In progress: Contains 8

Mailbox
Contains 2
Insert 3

Bottlenecks?
Concurrent message processing

- Possible issues?

- *List corruption*
  \[\Rightarrow\text{wrap message processing in an STM transaction.}\]

- *Altered message ordering*
  \[\Rightarrow\text{messages are not guaranteed to arrive in order because they are sent asynchronously}\]
Case 2: Coordinated message processing

A List Range (1..25)
Actor

...........

A List Range (100..125)
Actor

Global list sum
Global list sum – actor operations magnified

A List Range Actor

Local Actor state

2 → 5 → 6 → 8 → 9

1. Calculate local sum
2. Forward new partial sum
3. Block (for consistency)

Prev partial sum

New partial sum

Bottlenecks?
Case 2: Our solution

- Remove blocking; process other messages speculatively
  - ⇒ pre-commit the local coordinated transaction
  - ⇒ process other messages in a transaction speculatively
Our solution illustrated

A List Range Actor

Local Actor state

2 ➔ 5 ➔ 6 ➔ 8 ➔ 9

Messages
In progress

Speculatively process:
Contains 3

Saved: Partially committed
local coordinated transaction

Actor Mailbox
Contains 2

sum
Experimental settings

- Software: a modified Akka 2.10 distribution & Scala 2.10
- Hardware: 48-core AMD Opteron 6172 CPUs running at 2.1GHz
- Application: Stateful distributed sorted integer linked-list
Results

Execution time for sequential, concurrent, and non-blocking: write-dominated workload
Results

Execution time for sequential, concurrent, and non-blocking: read-dominated workload
Results

**Combined** concurrent & non-blocking execution
Results

Comparing with Habanero Scala

Habanero Scala
Summary

- By using speculation, we can achieve a higher message throughput in the Actor Model
- By using STM we guarantee that the Actor’s state is never corrupted
Questions?