Virtual Synchrony

Jared Cantwell
Review

• Multicast
• Causal and total ordering
• Consistent Cuts
• Synchronized clocks
• Impossibility of consensus
• Distributed file systems
Goal

• Distributed programming is hard
• What tools can make it easier?
• What assumptions can make it easier?

Distributed programming is hard!
Let’s go shopping!!!
The Process Group Approach to Reliable Distributed Computing

• Ken Birman
  – Professor, Cornell University

• ISIS
  – “toolkit mechanism for distributed programming”
  – Financial trading floors
  – Telecommunications switching
Virtual Synchrony

• Simplify distributed systems programming by assuming a *synchronous environment*

• Features:
  – Process Groups
  – Reliable Multicast
  – Fault Tolerance
  – Performance
Outline

• Problem / Motivation
• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
  – Virtual Synchrony
Outline

• Problem / Motivation
• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
  – Virtual Synchrony
Motivation

• Distributed Programming is hard
Difficulties

- No reliable multicast
- Membership churn
- Message ordering
- State transfers
- Failure atomicity
No Reliable Multicast

- UDP, TCP, Multicast not good enough
- *What is the correct way to recover?*
Membership Churn

- Membership changes are not instant
- How to handle failure cases?
Message Ordering

• Everybody wants it!
• How can you know if you have it?
• How can you get it?
State Transfers

- New nodes must get current state
- Does not happen *instantly*
- How do you handle nodes failing/joining?
Failure Atomicity

• Nodes can fail mid-transmit
• Some nodes receive message, others do not
• Inconsistencies arise!
Motivation Review

• *Distributed programming is hard!*

• No reliable multicast
• Membership churn
• Message ordering
• State transfers
• Failure atomicity
Outline

• Problem / Motivation

• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
  – Virtual Synchrony
Assumptions

• WAN of LANs
• Unreliable network
• Flow control at lowest layer
• Clocks not synchronized
• No partitions
  – CAP Theorem?
Failure Model

• Nodes crash
• Network is lossy
• Can’t distinguish difference
Outline

• Problem / Motivation
• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
  – Virtual Synchrony
Outline

• Problem / Motivation

• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
    • Model
    • Significance
    • Issues
  – Virtual Synchrony
Model

• Events (all or nothing)
  – Internal computation
  – Message transmission & delivery
  – Membership change
Model

- *Synchronous* execution

Ken’s Slides - 2006
Significance

• Multicast is always reliable
• Membership is always consistent
• Totally ordered message delivery
• State-transfer happens *instantaneously*
• Failure Atomicity
  – Multicast is a *single event*
Issues

- Discrete event simulator
- Is it practical?
- Impossible with failures
- Very expensive
  - System progresses in lock-step
  - Limited by speed of other members
Outline

• Problem / Motivation
• Solution (Virtual Synchrony)
  – Assumptions
  – Close Synchrony
  – Virtual Synchrony
Outline

• Virtual Synchrony
  – Asynchronous Execution
  – Virtual Synchrony
  – ISIS
  – Parallels
  – Benefits
  – Discussion
Asynchronous Execution

- Key to high throughput in distributed systems
- Only wait for responses (or too fast sends)
- Communication channel
  - Acts as a pipeline
  - Not limited by latency
- Not possible with Close Synchrony!!
Asynchronous Execution
Virtual Synchrony

- Close Synchrony + Asynchronous
- Indistinguishable to application
- So....when can synchronous execution be relaxed?
ISIS

- Communication Framework
- Membership Service
- VS primitives
  - ABCAST
  - CBCAST
ISIS

- Problem
  - *Crash and Lossy Network Indistinguishable*

- Solution:
  - Membership list
  - Nonresponsive or failed members are dropped
  - Only listed members can participate
  - Re-join protocol
  - *Does Membership exist in all distributed systems?*
ISIS

• Atomic Broadcast (ABCAST)
• No message can be delivered to any user until all previous ABCAST messages have been delivered
• Costly to implement
• ...But not everyone needs such strong guarantees
ISIS

- Causal Atomic Broadcast (CBCAST)
- Sufficient for most programmers
- Concurrent messages commute
- Weaker than ABCAST
When to use CBCAST?

Each thread corresponds to a different lock

- When any conflicting multicasts are uniquely ordered along a single causal chain
- .....This is Virtual Synchrony

Ken’s Slides - 2006
Parallels

- Logical time
- Replication in database systems
- Schneider’s *state machine approach*
- Parallel processor architectures
- Distributed database systems
Benefits

• Assume a closely synchronous model
• Group state and state transfer
• Pipelined communication (async)
• *Single event* model
• Failure handling
Discussion

- **Partitions**
- False positives
  - Most have them, VS admits it
- False negatives
  - Depend on a timeout
Summary

• Programming in distributed systems is hard
• Close Synchrony makes it easier
  – Costs too much
• Take asynchronous when you can
• Virtual Synchrony
  – Pipelined
  – Easy to reason over
Understanding the Limitations
of Causally and Totally Ordered Communication

• Authors
  – David Cheriton
    • Stanford
    • PhD – Waterloo
    • Billionaire
  – Dale Skeen
    • PhD – UC Berkeley
    • 3-phase commit protocol
The flaws of CATOCS

• Unrecognized causality
• No semantic ordering
• No Efficiency Gain (over State-level Techniques)
• No Scalability
Unrecognized Causality

- External communication is unknown
Unrecognized Causality

- Database is external entity
- Causal relation exists, but CATOCS misses it
No Semantic Ordering

• Serialization
  – Messages can’t be “group together”
  – Implementing eliminates CATOCS need

• Causal Memory
  – Solution: state-level logical clock
No Efficiency Gain

• Still need state-level techniques
• False causality
  – Reduces Performance
  – Increased Memory
• Message overhead
No Efficiency Gain

- What if m2 happened to follow m1, but was not causally related?
- CATOCS would make *False Causality*
No Scalability

• \( \approx \) quadratic growth of expected message buffering

• Rebuttal:
  – Worst case
  – Impractical use case
Summary

• CATOCS software is overkill
• Communication system doesn’t know everything
• Everything is better at the application level
Conclusions

• Distributed Programming is hard
• Close Synchrony
  – Too costly
• Virtual Synchrony
  – Limitations
• VS not perfect for all situations