Termination Detection for Diffusing Computations

Shankar

April 26, 2018

Overview

- Diffusing computation
  - distributed computation where each user is active or inactive
    - active: send/rcv msgs, become inactive
    - inactive: become active upon rcving a msg
  - starts with one active user, say `a0`
- Alg-level program of Dijkstra-Scholten algorithm for termination detection of diffusing computation
- Refine to await program that implements `TdChannel`
  - for case where only the sink is active initially

Outline

Termination detection: algorithm level

Termination detection: await-based program

Termination detection algorithm: overview

alg-level

- Distributed program `TdDiffusingDist`
  - starts a fifo channel and a system at each addr `j`
- Maintains a distributed out-tree rooted at `a0` over active users
  - exactly one directed path from `a0` to every active user
  - path may go via non-leaf inactive users
  - no other edges, ie, no undirected cycle
- Creates `[j,k]` when non-tree `k` rcvs a `j`-msg
- Deletes `[j,k]` when `k` is a leaf and inactive
- `a0` detects termination when it is inactive and a leaf
- User finds out it is a leaf via acks to user msgs

Termination detection algorithm

alg-level

- Systems, each with a user, attached to a fifo channel
- Users exchange msgs, which systems relay over fifo channel

- System messages
  - data msg [DAT, sender addr, user msg]
  - ack msg [ACK]

- System j vars
  - active: initially true for a0, o/w false
  - engager: initially a0 for a0, o/w null

    // "up-stream" neigbor if j in the tree, o/w null
  - unAcked: initially 0

    // # of unacked outgoing data msgs

System j rules – 1

- only if active = true:
  - active ← false
- only if active
  - send [DAT,j,umsg] to k
  - unAcked++
- receive [DAT,k,dmsg]:
  - active ← true
  - if engager = null
    engager ← k
  - else
    send [ACK] to k

System j rules – 2

- receive [ACK]
  - unAcked--
- Disengage

  only if (not active and unAcked $=$ 0 and engager $\neq$ null)
  - if j $=$ a0

    signal termination
  - else

    send [ACK] to engager

    engager $\leftarrow$ null

- Assumptions
  - rules are atomic
  - weak fairness for disengage

Analysis auxiliary quantities

- numDAT(j): # data msgs in transit outgoing from j
- numACK(j): # ack msgs in transit incoming to j
- termination: forall(j: not j.active and numDAT(j) $=$ 0)
- eNodes: set(j: j.engager $\neq$ null) // engaged nodes
- eEdges: bag([k.engager, k]: k $\neq$ a0, k.engager $\neq$ null)
  // engagement edges
- eGraph: [eNodes, eEdges] // engagement digraph

Assertions to be proved

- Safety

  A_1 : Inv `(a0.unAcked = 0 and not a0.active)` $\Rightarrow$ `termination`

- Progress

  A_2 : `termination` *leads-to* `(a0.unAcked = 0 and not a0.active)`

Proof of A_1

- Intermediate predicates

B_1 : eGraph is an out-tree rooted at a0

B_2 : `j.unAcked = numDAT(j) + numACK(j) + sum([j,k]: [j,k] in eEdges)`

B_3 : `j.engager = []` $\Rightarrow$ (not `j.active` and `j.unAcked = 0`)

- *Inv* B_1–B_3: B_1–B_3 satisfies invariance rule
- B_1–B_3 implies A_1's predicate
- hence A_1 holds

Proof of A_2

A_2 : `termination` *leads-to*
`(a0.unAcked = 0 and not a0.active)`

- Assume `termination` // all inactive, no data msgs in transit
- Assume eEdges is not empty
  - so there is a leaf node `j`
  - `j` has no outgoing data msgs or incoming edges
  - `j`'s incoming acks are eventually rcvd
  - so `j.unAcked` becomes `0` and stays so
  - so `j` sends an ack to its engager and leaves the tree
- Eventually eEdges is empty and `a0.unAcked` is `0`

Outline

Termination detection: algorithm level

Termination detection: await-based program

Termination detection program

- Distributed program `TdDiffusingDist(ADDR, a0)`
  // implements `TdChannel` for only a0 initially active
  - $\{c_j\} \leftarrow$ start `FifoChannel(ADDR)`
  - for j in ADDR
    $v_j \leftarrow$ start `TdDiffusing(ADDR, j, a0,` c_j`)`
  - return $\{v_j\}$

- `TdDiffusing`: await program, refines alg-level system
  - input fns: `tx`, `rx`, `inactive`, `isTerminated` (only at a0)
  - output calls: `tx`, `rx` of channel access system

Program `TdDiffusing`: overview

- Parameters
  - `ADDR`, local addr `j`, sink addr `a0`, channel access system c_j
- Input fns (called by user)
  - `tx(k,msg)`
  - `rx()`
  - `inactive()` // indicates user inactive
  - `isTerminated()` (only at `a0`) // return only if termination
- Local fn `doRx()`, executed by local thread
  - rcvs msg from channel, update td state
  - add user msg (if any) to a buffer // user rcvs from buffer
    // it's part of user wrt td state

- Main
  - active ← (j = a0)
  - engager ← if (j = a0) a0 else null
  - unAcked ← 0
  - rxq ← [] // buffer for rcvd user msgs
  - startThread (doRx()) // rcvs msgs from channel
  -
- input mysid.tx(k, msg)
  - await (true)
    - unAcked++
    - c_j.tx(k, [DAT, j, msg])
  - return

Program TdDiffusing – 2

- input mysid.rx()
  - await (rxq.size > 0)
    - msg ← rxq[0]
    - rxq.remove()
  - return msg
  - // return [msg,k]

- input mysid.inactive()
  - await (true)
    - if rxq = []

      active ← false

      if ($j \neq a0$ and unAcked = 0)

        c_j.tx(engager, [ACK]) // disengage

        engager ← null

Program TdDiffusing – 3

- function doRx()   // executed by a local thread

```
while true
  msg ← c_j.rx()    // ia {msg is [DAT, k, msg], [ACK]}
  await true
```

  - if msg = [DAT, k, msg]
    - rxq.append(msg)
    - active ← true
    - if (engager = null) engager ← k
      else c_j.tx(k, [ACK])
  - else if msg = [ACK]
    - unAcked --
    - if ($j \neq$ a0 and unAcked = 0 and not active)
      c_j.tx(engager, [ACK])   // disengage
      engager ← null

Program TdDiffusing – 4

- input mysid.isTerminated()
  - ia {j = a0} // only at a0
  - await not active and unAcked = 0
  - return

- atomicity assumption {awaits}
- progress assumption {wfair threads}

Analysis: overview

- To prove: `TdDiffusingDist(ADDR, a0)`
  implements `TdChannel(ADDR, a0, a0)`

- Usual steps
  - define program of implementation $\{v_j\}$ and service inverse `si`
  - identify effective atomicity breakpoints
  - obtain assertions
  - prove program satisfies assertions // easy given ⋆

- Assertions deal with two issues
  - fifo channel
  - termination detection

Analysis: fifo channel assertions

- `[msg,k]` returned by `j.rx` next in fifo order from k
  - recall fifo channel rx has internal param sender-addr k
  - so augment `j.rx` return (and `j.rxq` entries) with k

  Proof: $\texttt{si.rxh}_{\texttt{k,j}} \circ$ (rxq k-entries) $=$ ($\texttt{chan.rxh}_{\texttt{k,j}}$ data entries)

- msg in transit is eventually rcvd if `j.rx` ongoing

  Proof: msg enters `j.rxq` (channel prog), then user (await fairness)

- ongoing `j.tx` eventually returns

  Proof: `j.tx` is non-blocking, await fairness

Analysis: termination detection assertions

- `a0.unAcked` $=$ `0` and not `a0.active` implies termination
- termination *leads-to* `a0.unAcked` $=$ `0` and not `a0.active`
- Proof: Follow from (similar) alg-level A_1–A_2 subject to
  - `j.active` $\Leftrightarrow$ (`si.active[j]` or `j.rxq` $\neq$ `[]`)