Distributed Shared Memory Service

Shankar

June 1, 2014

Overview: Distributed shared memory – 1

- Memory shared by multiple threads on different computers
  - implemented via msg-passing
  - may also use hardware support // eg, in caches

- Provides read and write operations at the minimum
  - *val* ← read(*addr*)
  - void ← write(*addr*, *val*)

- Basic approach to implementation
  - divide memory into pages
  - "move" (a copy of) page to the user currently accessing it

- Fundamental issue:
  - when does a user's write become visible to another user
  - traditional memory: write becomes visible immediately
  - distributed memory: trade-offs

Overview: Distributed shared memory – 2

- Traditional "sequential" memory
  - atomic reads and writes
  - read returns last write's value
  - poor performance on distributed platform
- Distributed memory
  - allow reads to return earlier values
  - improved performance, but weaker correctness
  - sequential-consistency: equivalent to sequential wrt correctness
  - release consistency: weaker than sequential
  - ...
- Define sequentially-consistent memory service
  - `DsmSeqConService(ADDR, PNO, PVAL)`

DsmSeqConService – 1

- Parameters
  - ADDR: addresses
  - PNO: page numbers
  - PVAL: possible values of a page
- Input functions at addr j
  - j.read(pn) and j.write(pn, pv)
- Main
  - Seq $\alpha \leftarrow$ [] // seq of "write-call" and "read-ret" entries
    - [WCALL, j, pn, pv] // write-call
    - [RRET, j, pn, pv] // read-ret
  - return $\{v_j \leftarrow$ sid()$\}$ // access system at j

DsmSeqConService – 2

- Below h is a seq of WCALL and RRET entries

- Helper fn lastWrite(h, pn) // last val written to pn in h

  xh ← [[WCALL, ·, pn, .] in h]

  return xh.last.pv

- Helper fn sequential(h) // true iff every read returns last write

  return forall(i in h.keys, h_i = [RRET, . ,pn, pv]:

  h_i = lastWrite($h_{0..i-1}$, pn))

- Helper fn seqConsistent(h)

  return forsome(Seq xh:

  sequential(xh) and

  forall(j: [[., j, ., .] in xh] = [[., j, ., .] in h]))

DsmSeqConService – 3

- v_j.read(pn)
  - ic { no ongoing v_j.read(.) or v_j.write(.) }
  - output pv

    oc { seqconsistent($\alpha \circ$ [[RRET, j, pn, pv]])}

    α.append([RRET, j, pn, pv]

    return pv

- v_j.write(pn, pv)
  - ic { no ongoing v_j.read(.) or v_j.write(.) }

    α.append([RRET, j, pn, pv]
  - oc { true }

    return

DsmSeqConService – 4

- atomicity assumption: input parts and output parts
- progress assumption:
  - ongoing j.read(.) *leads-to* no ongoing j.read(.)
  - ongoing j.write(.) *leads-to* no ongoing j.write(.)

Coherent shared memory

- Remote sequential memory // accessed via fifo channel
- Equivalent to seq-consisent memory with additional constraint
  - for every read or write return r:
    permutation must not move any part after r to before r
- Need hardware support to implement sequentially-consistent but not coherent