A Single-Copy Distributed Shared Memory

Shankar

June 3, 2014

Outline

Single-Copy Distributed Shared Memory Implementation

Proving the implements conditions

Overview

- Program DsmSCopyDist:
  - implements coherent dsm using object-transfer service
  - starts obj-transfer service over given addrs
  - starts DsmSCopy system at every addr of obj-transfer service
- Each DsmSCopy system holds a subset of pages for its user
- Initially, pages are partitioned amongst DsmSCopy systems
- When user attempts access to a page
  - if page is not local, use obj-transfer service to obtain page
  - if page is local, access page as usual

Distributed program

- program DsmSCopyDist (ADDR, PNO, PVAL, $\{\text{initPages}_j\}$)

  // initPages_j: initial page numbers at addr j

  $\{w_j\} \leftarrow$ start ObjTransferService (ADDR, PNO, PVAL, initPages)

  for j in ADDR

  $v_j \leftarrow$ start DsmSCopy (PNO, PVAL, initPages_j, w_j)

  return $\{v_j\}$

- Input fns: v_j.read(pn), v_j.write(pn, pv)

- Implements DsmSeqConService (ADDR, PNO, PVAL)

Program DsmSCopy – 1

- Parameters: PNO, PVAL, initPages_j, w_j // note: no addrs
- Input fns: mysid.read(pn), mysid.write(pn, pv)
- Local fns: doRxReq() // rcvs reqs from obj-xfr service
- Output calls: w_j.acq(pn), w_j.rel(pn, pv), w_j.rxReq()
- Main
  - pgVal: map <PNO, PVAL> // pn-entry iff page pn local
    $\text{pgVal}_{pn} \leftarrow 0$, for pn in initPages_j
  - startThread(doRxReq())

- input mysid.read(pn):
  await (true)
    if (pgVal has pn entry)
      return $pgVal_{pn}$
  - tpv $\leftarrow$ w_j.acq(pn)
  await (true)
    $pgVal_{pn} \leftarrow$ tpv
    return $pgVal_{pn}$

- input mysid.write(pn, pv):
  await (true)
    if (pgVal has pn entry)
      $pgVal_{pn} \leftarrow pv$
      return
  - $tpv \leftarrow w_j$.acq(pn)
  await (true)
    $pgVal_{pn} \leftarrow tpv$
    $pgVal_{pn} \leftarrow pv$
    return

- function doRxReq():
  while (true)
  - pn $\leftarrow$ w_j.rxReq()

  ic { pn in PNO }
  await (pn-entry in pgVal)
  w_j.rel(pn, $pgVal_{pn}$)
  remove pn-entry from pgVal

- atomicity assumption: await
- progress assumption: weak fairness for all threads

Outline

Single-Copy Distributed Shared Memory Implementation

Proving the implements conditions

Analysis overview

- To prove: `DsmSCopyDist` implements `DsmSeqConService`
- Define program `Z` of implementation and service inverse `si`
- Identify effective atomicity breakpoints // ●
- Identify assertions A to hold
- Prove `Z` satisfies A

- Parameters:
  - ADDR, PNO, PVAL, initPages

- DsmSCopy at j:

  j.pgVal

  startThread(doRxReq())

- DsmSeqConServiceInverse:

  ic {···}

  $\alpha \leftarrow$ [] // wcalls, rrets

- input j.read(pn):
  await (true)
  if ($pgVal_{pn}$ exists)
  b1: return $pgVal_{pn}$
  - $tpv \leftarrow w_j$.acq(pn)
  await (true)
  $pgVal_{pn} \leftarrow tpv$
  b2: return $pgVal_{pn}$

- doRead(j,pn)
  oc { no ongoing
  j.read(.) or j.write(.)
  pv $\leftarrow$ j.read(pn)
  ic { seqConsistent(
  $\alpha \circ$ [[RRET, j, pn, pv]])}
  α.append([[RRET, j, pn, pv]])}

- input j.write(pn, pv):
  await (true)
  if ($pgVal_{pn}$ exists)
  $pgVal_{pn} \leftarrow$ pv
  b3: return
  b4: • tpv $\leftarrow w_j$.acq(pn)
  await (true)
  $pgVal_{pn} \leftarrow$ tpv
  $pgVal_{pn} \leftarrow$ pv
  b5: return

- doWrite(j,pn)
  oc { no ongoing
  j.read(.) or j.write(.)
  α.append([[WCALL, j, pn, pv]])}
  j.write(pn, pv)
  ic { true }

- function doRxReq(): ...
- atomicity assumption: ...
- progress assumption: ...

- atomicity assumption: ...
- progress condition: ...

- ObjTransferService(ADDR, PNO, PVAL, initPages)

Outline

Assertions to establish

- To satisfy service, need to prove Z satisfies A_1

  A_1 : Inv thread at `si.doRead(j,pn).ic`

  $\Rightarrow$ `seqConsistent(` α $\circ$ `[[RRET, j, pn, pv]])`

- A_1 equivalent to $Inv\ B_1$ // wcall preserves seq consistency

  B_1 : `seqConsistent(` α `)`

Proof – 1

- Define auxiliary var β st Z satisfies Inv B_2–B_3

  B_2 : forall(j: [[., j, ., .] in β] = [[., j, ., .] in α]))

  B_3 : sequential(β)

- Candidate: $\beta = \beta_S \circ \beta_T$ // stable ∘ transient
  - when [.,j,pn,pv] is appended to α
    - append it to β_S if $\mathtt{j.pgVal}_{\mathtt{pn}}$ exists // at b1, b2, b3
    - append it to β_T o/w // at b4
  - when j.write(pn,pv) call becomes unblocked
    - remove its entry from β_T and append to β_S // at b5

- Sufficient to prove Inv B_2–B_3

Proof – 2

- Requires proving obj-xfr service is used correctly // o/w fault

  B_4 : thread at $\mathtt{w_j.acq(pn)} \Rightarrow$ no ongoing $\mathtt{w_j.acq(pn)}$

  B_5 : thread at $\mathtt{w_j.rel(pn, pv)} \Rightarrow$ (pn in $\mathtt{w.objs_j}$ and in $\mathtt{w.reqs_j}$)

  B_6 : thread at $\mathtt{w_j.rxReq()} \Rightarrow$ no ongoing $\mathtt{w_j.rxReq()}$

- Proving Inv B_2–B_6 is straightforward
  - proving Inv B_4–B_6 is trivial
  - some helpful predicates are given next

Some helpful predicates

- Possibly helpful predicates in proving Inv B_2–B_6

B_7 : forall pn, exactly one of the following hold:
  - forone(j: $\mathtt{pgVal}_{\mathtt{pn}}$ exists)
  - forall(j: no $\mathtt{pgVal}_{\mathtt{pn}}$ exists) and (w.$\mathtt{val}_{\mathtt{pn}}$ exists)

B_8 : $\mathtt{val}_{\mathtt{pn}}$ exists $\Rightarrow$ $\mathtt{val}_{\mathtt{pn}} = \mathtt{lastWrite}(\beta_S, \mathtt{pn})$

B_9 : j.$\mathtt{pgVal}_{\mathtt{pn}}$ exists $\Rightarrow$ $\mathtt{val}_{\mathtt{pn}} = \mathtt{lastWrite}(\beta_S, \mathtt{pn})$

Outline

Proof of progress: A_2–A_3

- Obj-xfr service progress holds because it is used correctly
- Given this, proof o fA_2 and A_3 is straightforward
- See text for details

Assertions to establish

- To satisfy service, need to prove Z satisfies A_1–A_2

  A_2 : ongoing `j.read(.)` *leads-to* not ongoing `j.read(.)`

  A_3 : ongoing `j.write(.)` *leads-to* not ongoing `j.write(.)`

Proof

- Obj-xfr service progress holds // coz Z uses service correctly

P_3 : ongoing w_j.rel(.) *leads-to* no ongoing w_j.rel(.)

P_4 : (... *leads-to* ...) $\Rightarrow$
(pn $\in$ w.objs$_j$ & ongoing w_k.acq(pn)) *leads-to* pn $\in$ w.reqs$_j$

P_5 : (... *leads-to* ...) $\Rightarrow$
ongoing w_j.acq(pn) *leads-to* no ongoing w_j.acq(pn)

- Thread in j.doRxReq() always waits for a req and releases the requested page

P_2 : pn in w.reqs$_j$ *leads-to* not pn in w.reqs$_j$

- P_2 implies P_3–P_5.lhs.
- P_3–P_5.rhs implies A_2–A_3