A Multi-Copy Distributed Shared Memory

Shankar

June 5, 2014

Multi-Copy Distributed Shared Memory Implementation

Proving the implements conditions

Overview

- Multi-copy version of single-copy dsm implementation
- For each page, maintain ≥ 1 copies at different addrs
  - 1 write copy and ≥ 0 read copies
  - all the copies have the same value
  - copyset: addrs of read copies // accompanies write copy
- Read from write copy or read copy
- Write to write copy only when no read copy exists
- To read or write without local copy
  - acquire write copy // leave read copy at old addr
  - if write, ask copyset sites to delete their copies // invalidate
- Uses two services: object-transfer and fifo channel

Distributed program

- program DsmMCopyDist (ADDR, PNO, PVAL, $\{initPages_j\}$)

  // $initPages_j$: initial page numbers at addr j

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

  $\{c_j\} \leftarrow$ start FifoChannel (ADDR)

  for j in ADDR

  $v_j \leftarrow$ start DsmMCopy (PNO, PVAL, $initPages_j$, j, w_j, c_j)

  return $\{v_j\}$

- Input fns: v_j.read(pn), v_j.write(pn, pv)
- Implements DsmSeqConService (ADDR, PNO, PVAL)
- Uses ObjTransferService to transfer pages
- Uses FifoChannel to send invalidate reqs and acks

Program DsmMCopy – 1

- Parameters: PNO, PVAL, $\texttt{initPages}_j$, w_j, j, c_j
- Input fns: mysid.read(pn), mysid.write(pn, pv)
- Local fns
  - doRxReq() // rcvs reqs from obj-xfr service
  - doRx // rcv invalidate msgs from fifo channel
- Output calls
  - w_j.acq(pn): returns [pv, cs] // [page value, copyset]
  - w_j.rel(pn, [pv, cs])
  - w_j.rxReq(): returns page number
  - c_j.tx(k, msg): send msg // [IREQ, j, pn], [IACK, j, pn]
  - c_j.rx(): rcv msg

- Main
  - Following exist iff page n copy locally present
    - $\texttt{pgval}_n$: value of page n // initially 0 for n in $\texttt{initPages}_j$
    - $\texttt{wcopy}_n$: true iff write copy n
    - $\texttt{invaldng}_n$: true iff write-copy n is being invalidated
    - $\texttt{cpset}_n$: copyset of n iff n write-copy
  - startThread(doRxReq())
  - startThread(doRx())
- atomicity assumption: await
- progress assumption: weak fairness for all threads

- input mysid.read(n):
  await (true)
    if ($pgVal_n$ exists)
      return $pgVal_n$
  - $[tpv, cs] \leftarrow w_j.acq(n)$
  await (true)
    $pgVal_n \leftarrow tpv$
    $wcopy_n \leftarrow$ true
    $invaldng_n \leftarrow$ false
    $cpset_n \leftarrow cs$
    return $pgVal_n$

- input mysid.write(n, pv):

```
acqpage ← true
await (true)
  if ( pgVal_n exists and wcopy_n )
    if ( cpset_n empty )
        pgVal_n ← pv
        return
    else
        invaldng_n ← true
        acqpage ← false
// acqpage if no write copy, o/w need invalidating
if (acqpage)
  • [tpv, cs] ← w_j.acq(n)
// mysid.write(n, pv) continued
```

```
// mysid.write(n, pv) continuing
await (true)
  if (acqpage)
    pgVal_n ← tpv
    invaldng_n ← wcopy_n ← true
    cpset_n ← cs \ set(j)
  // send invalidate requests
  for (k in cpset_n)
    c_j.tx( k, [IREQ, j, n] )
// wait for copyset to empty, then write
await (cpset_n empty)
  pgVal_n ← pv
  invaldng_n ← false
  return
// end of mysid.write
```

- // rcv pn n from obj-xfr service; release n

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

    await ($pgVal_n$ exists and not $invaldng_n$)

    w_j.rel(n, [$pgVal_n$, union($cpset_n$, set(j))])

    $wcopy_n \leftarrow$ false

- // rcv IREQ/IACK msgs from fifo channel

```
function doRx():
 while (true)
   • msg ← c_j.rx()
    if (msg = [ IREQ, k, n ])
      // read copy of n exists or will soon exist
      await (pgval_n exists)
        delete pgval_n
        c_j.tx(k, [ IACK, j, n ])
    else if (msg = [ IACK, k, n ])
      // write copy of n exists and being invalidated
      await (true)
        cpset_n.remove(k)
```

Multi-Copy Distributed Shared Memory Implementation

Proving the implements conditions

Analysis overview

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

- Analysis mirrors that of single-copy implementation
- Auxiliary var β defined there also works here