On the Design and Implementation of DSM-Threads

Frank Mueller

Institut für Informatik
Humboldt-Universität zu Berlin
10099 Berlin (Germany)

e-mail:
mueller@informatik.hu-berlin.de

WWW:
`http://www.informatik.hu-berlin.de/~mueller`

Overview

- distributed threads atop POSIX threads
- shared memory via DSM
- migrate concurrent ⇒ distributed
- retain programming model (shared memory)
- design and implementation

Motivation

- problem w/ shared-memory multiprocessors (SMPs):
  - bus is bottleneck
- problem w/ distributed programming:
  - absence of global state
  - centralized (server-client) w/ bottleneck
  - decentralized w/ new and complex algorithms
- DSM: best of both worlds (NUMA)
- exploit processing power of existing network of workstations
- bottleneck is network throughput (some improvements, but ...)

Design Goals

- portability
- compatibility
- communication issues
- extensibility and tuning
- heterogeneous support
- modular design
- multi-paradigm support

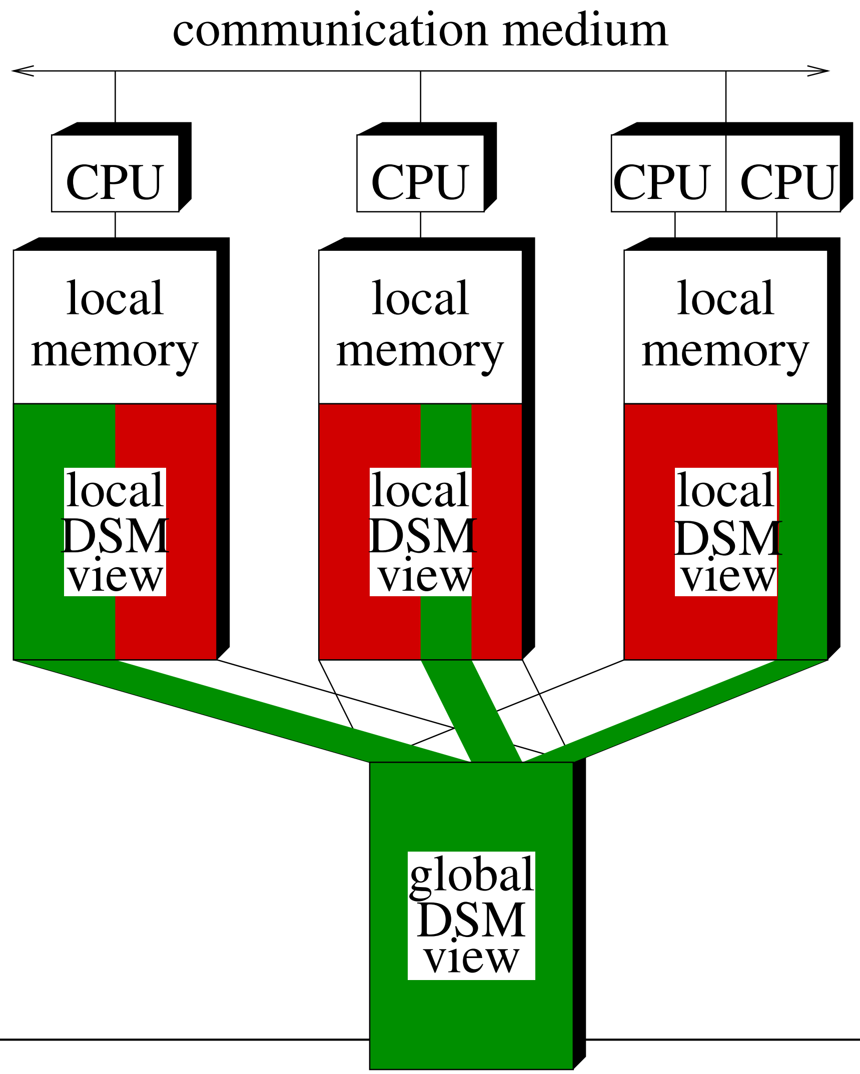
- portability:
  - no modifications to compiler, OS, HW
  - use standards (POSIX), Gnu utilities
- compatibility:
  - interface close to Pthreads (syntactically and semantically)
  - minimal modifications to source programs when migrating from Pthreads to DSM-Threads (toward "backwards compatibility")
- communication issues:
  - least possible communication overhead (since communication may be slow)
  - decentralized distributed algorithms (avoid bottlenecks)
- extensibility and tuning:
  - simple to advanced interfaces to fine-tune applications (for strait ports to expert users)
  - extensible interface (for future enhancements)
- heterogeneous support:
  - transparent handling of heterogeneous issues
  - use standards for data en-/decoding
-
- modular design:
  - independent subsystems (e.g. communication, simple interface, advanced interface)
  - easily replaced by new protocols for other media (e.g. ATM)
- multi-paradigm support:
  - DSM for regular data and object-oriented encapsuling
  - support via different consistency protocols

Task Model

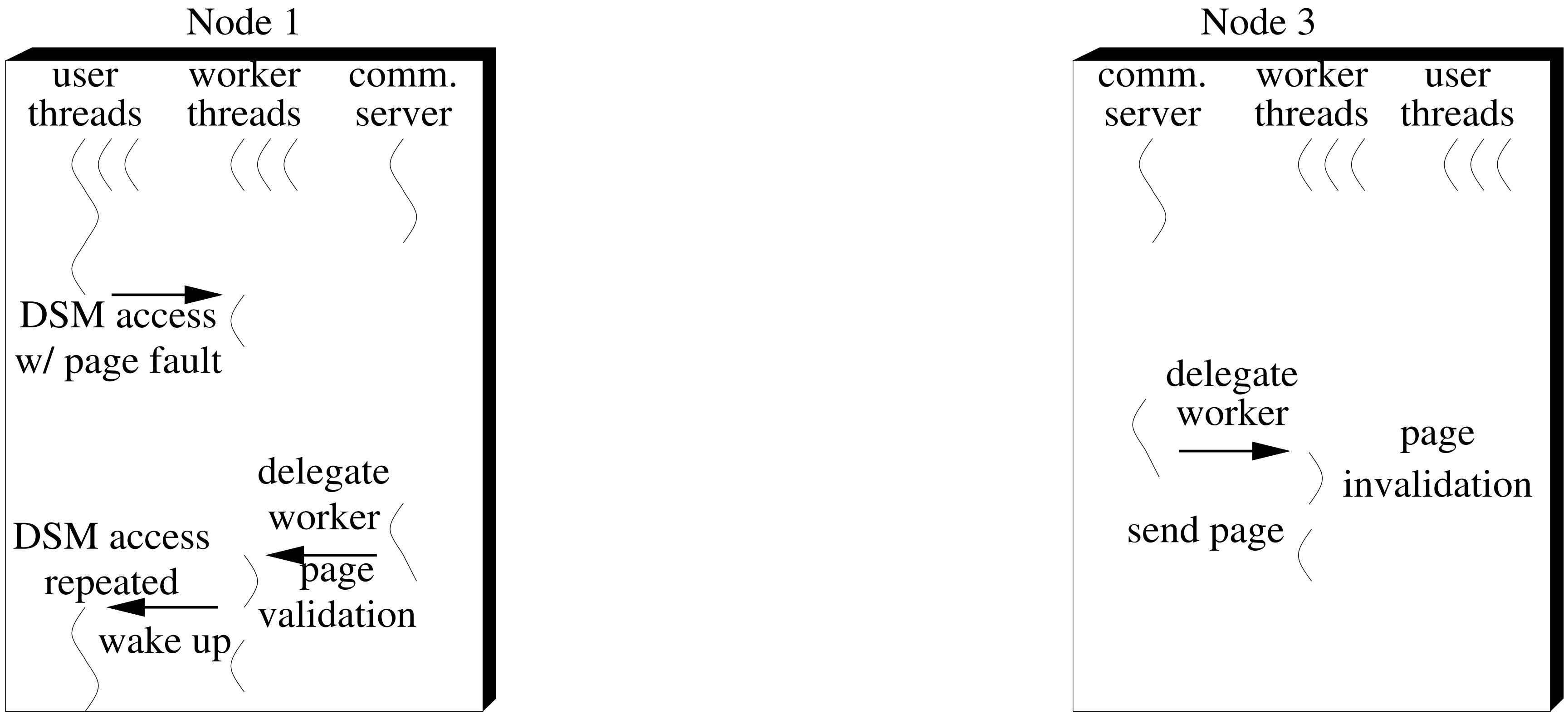
- node = (host, process)
- process: DSM user / system threads, Pthreads
- system threads
  - communication server
  - worker threads
- responsiveness
- page fault model

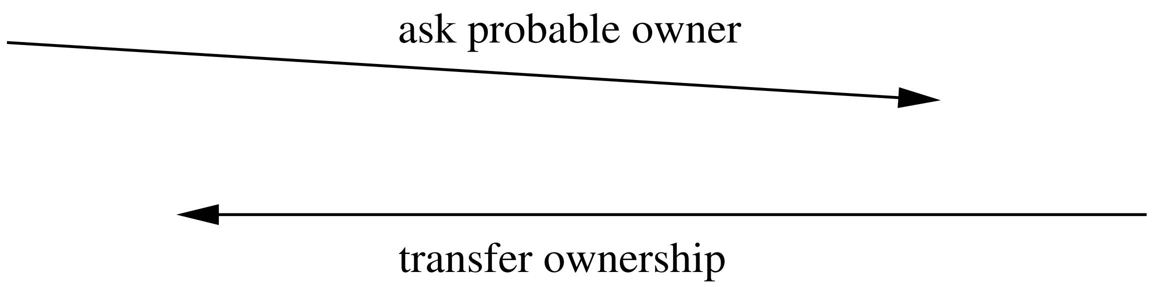
- node = (host, process)
- process: DSM user / system threads, Pthreads
- communication server
  - highest prio
  - open comm. channel, wait for msgs
  - decode msg header
  - act on msgs or delegate
- worker threads
  - process msg
  - handle page faults
  - serve as new user thread
- responsiveness
  - $|$ workers $| > |$ msg types $|$
  - allocate new worker if
    - no other worker handles this msgs type yet OR
    - $|$ idle workers $| > |$ unhandled msg types $|$
- page fault in user thread
  - insufficient access right to datum
  - page fault handler to active worker
  - worker obtains proper access rights
  - ask comm. server of other node
  - local comm. server receives data / access rights
  - worker updates data and access right
  - worker reactivates user thread

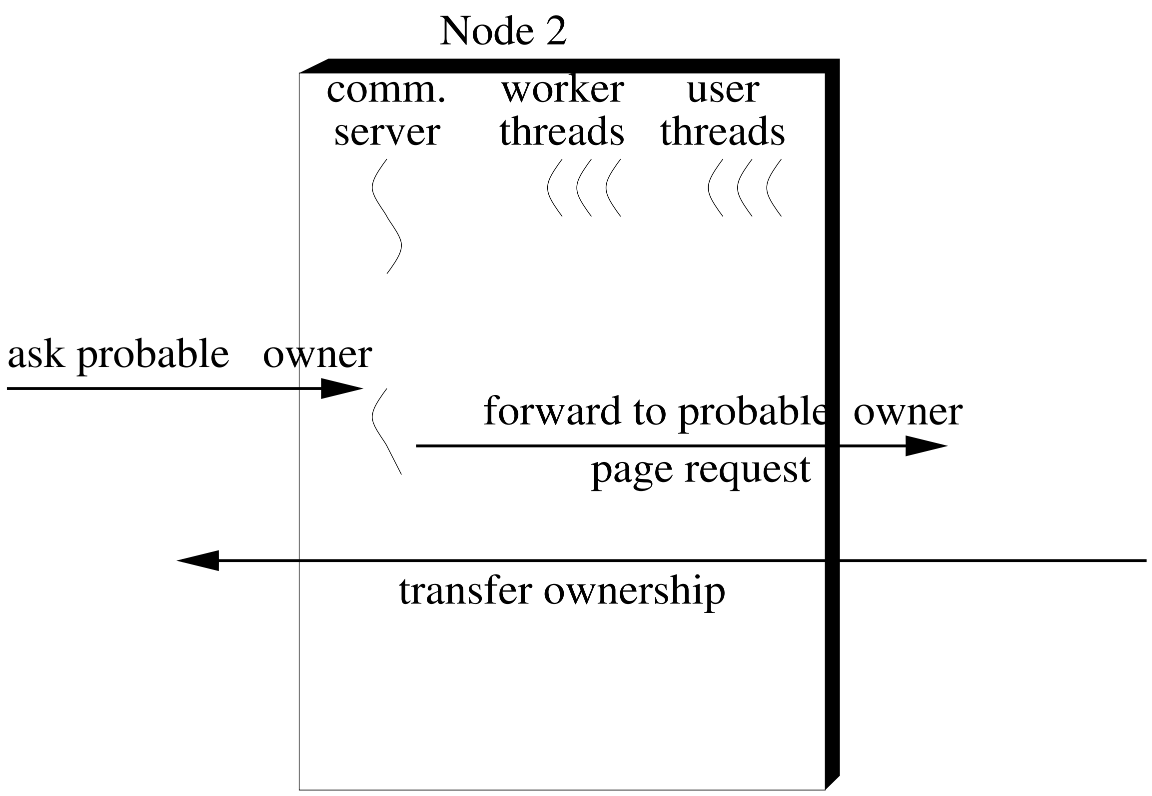
DSM Model

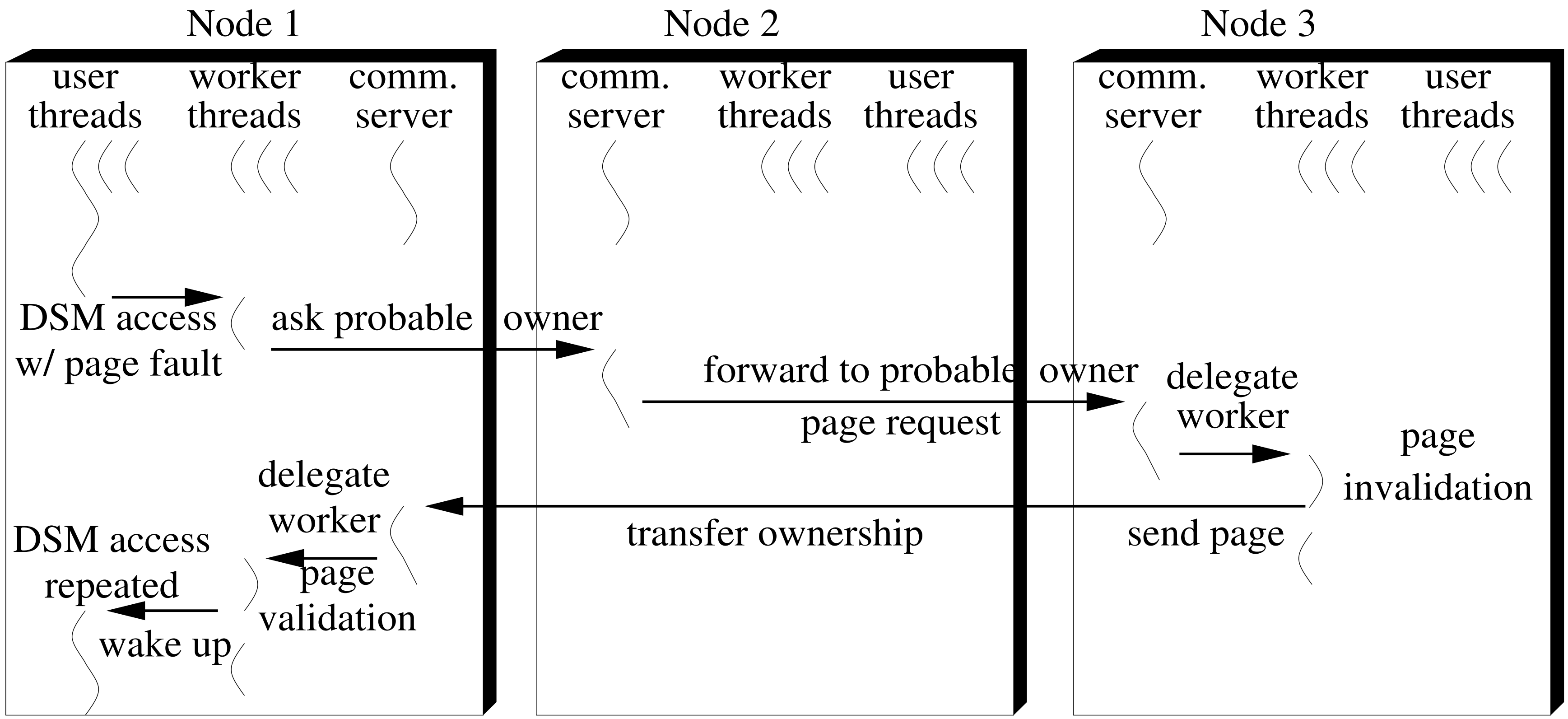

communication medium
CPU
CPU
CPU
CPU
local memory
local memory
local memory
local DSM view
local DSM view
local DSM view
global DSM view

Page Fault Model




ask probable owner
transfer ownership


Node 2
comm. server
worker threads
user threads
ask probable owner
forward to probable owner
page request
transfer ownership


Node 1
Node 2
Node 3
user threads
worker threads
comm. server
comm. server
worker threads
user threads
comm. server
worker threads
user threads
DSM access w/ page fault
ask probable owner
forward to probable owner
page request
delegate worker
page invalidation
transfer ownership
send page
delegate worker
DSM access repeated
page validation
wake up

DSM Data Identification

- page-based consistency
  - SRNW
  - SRSW
  - MRSW
- data-based consistency
  - MRMW
- no pointers to or within DSM data
- DSM data must be initialized
- no compiler or OS modifications
- ELF may facilitate issues

- page-based consistency
  - SRNW: single-reader-no-writer (read-only)
  - SRSW: single-reader-single-writer
  - MRSW: multi-reader-single writer: improved dynamic distributed manager algorithm by Li and Hudak
- data-based consistency
  - MRMW: multi-reader-multi-writer (entry consistency)
  - implements MRSW for a set of data
  - viewed as MRMW on the page level since multiple data on one page
- DSM data must be initialized
- no compiler or OS modifications
- ELF may facilitate issues (initialized + uninitialized data)

Heterogeneous Support

- protocol header to identify compatibility
- data encoding for DSM data (if necessary) via XDR
- assumes same sequence of DSM data layout
  - same DSM pages and DSM data index
  - not necessarily same address
- data alignment problems:
  - problem 1: different pages for same data
  - $\Rightarrow$ preprocessor forces page feeds
  - problem 2: large data structures $>$ one page
  - $\Rightarrow$ treat consecutive pages as one entity

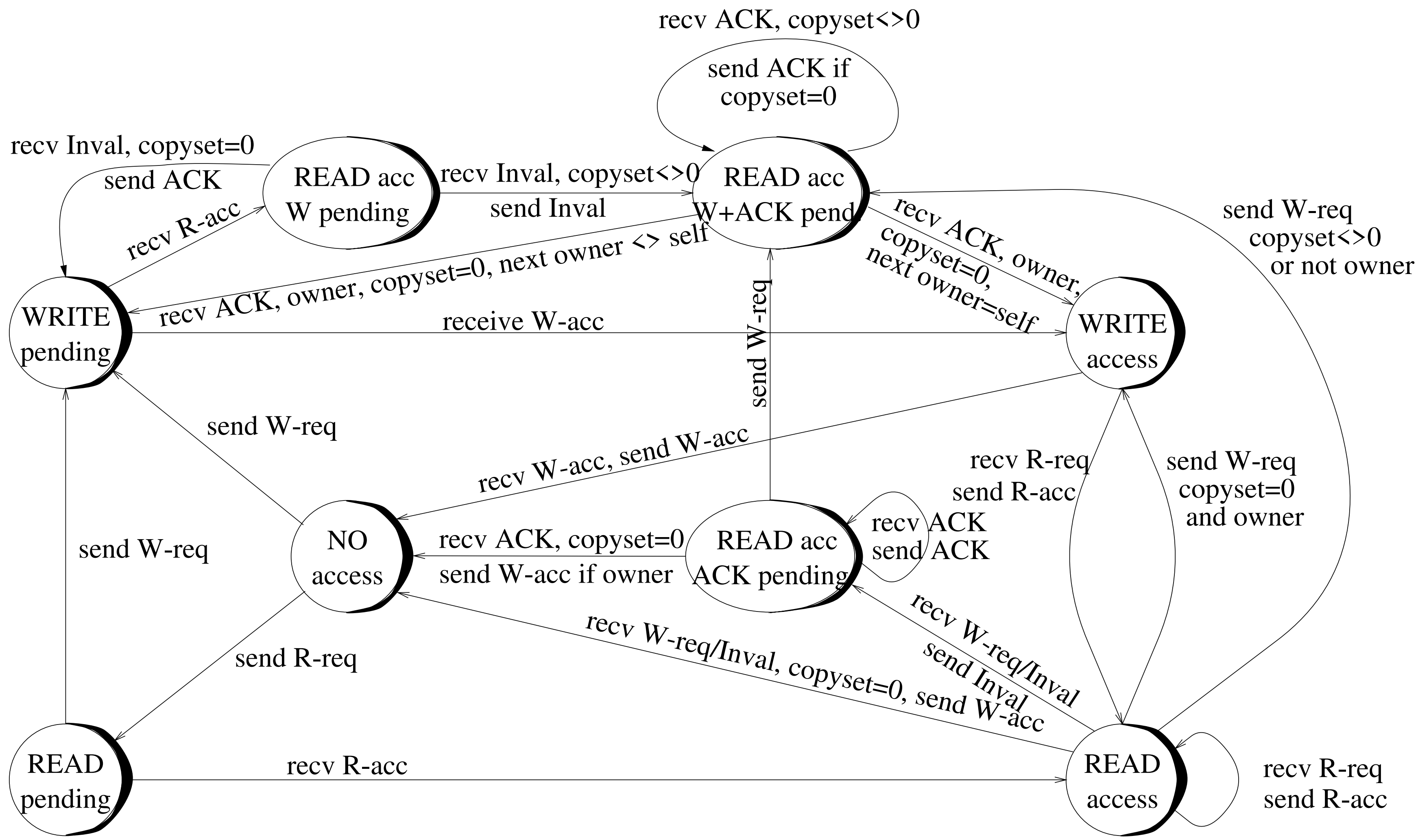
- protocol header to identify compatibility
- data encoding for DSM data (if necessary) via XDR
- assumes same sequence of DSM data layout
  - same DSM pages and DSM data index
  - not necessarily same address
- data alignment problems:
  - problem 1: different pages for same data
  - $\Rightarrow$ preprocessor forces page feeds
  - sample each architecture, track max. space in preprocessor
  - modest amount of unused space for some, all used for one architecture
  - problem 2: large data structures $>$ one page
  - $\Rightarrow$ treat consecutive pages as one entity
  - on page fault, request entire page range

Asynchronous Access Model

- multiple threads per node $\Rightarrow$ concurrency
- responsiveness
- idea: send requests asynchronously
- $\Rightarrow$ states of DSM object
- other advantages:
  - DSM object read accessible while ACKs pending
  - remote requests forwarded / stored while local requests pending
  - direct read $\leftrightarrow$ write access transitions
- invalidation only initiated by father of copyset

- multiple threads per node ⇒ concurrency
- responsiveness (do not block other threads in process)
- idea: send requests asynchronously (don't wait for answer)
- ⇒ states of DSM object
  - applicable to page-based MRSW and data-based MRMW (EC)
- other advantages:
  - DSM object read accessible while ACKs pending
  - remote requests forwarded / stored in distr. Q while local requests pending (omitted in transition diagram)
  - direct read ↔ write access transitions
    - lower access W → R on read request
    - raise access R → W if copyset empty
- invalidation only initiated by father of copyset (o/w race)
- more in paper (state transition table, actions for consistency protocol)

State Transitions of DSM Objects

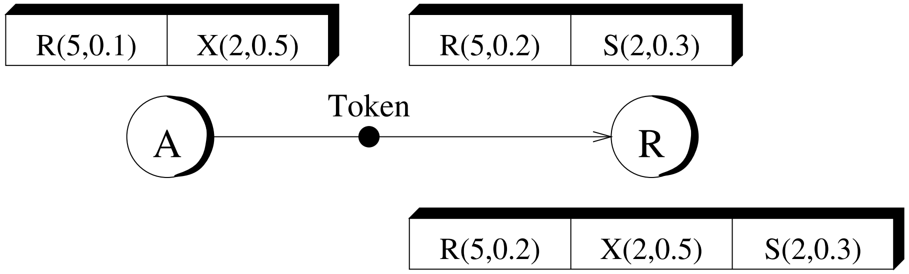


Prioritized Protocol

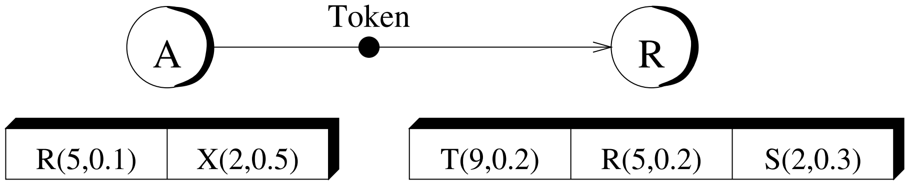
- POSIX requires strict priority scheduling
- new protocol for token-based distributed mutual exclusion
- probable owners annotated with priority levels
- set of local queues stores requests
- stored requests age
- on token passing, sender's queue merged with receiver's
- merging follows
  - strict priority ordering, then
  - descending age
- if head(Q) = self $\Rightarrow$ use token (mutex) else fwd to head(Q)
- no global logical clocks, no timestamps, no broadcast
- $\Rightarrow$ new protocol, same performance as best unprio. prot.
- $\Rightarrow$ guide thread distribution

- POSIX requires strict priority scheduling (including mutex protocols), applications:
  - real-time support (to finish within deadline, avoid priority inversion when resources are blocked)
  - tuning (to reduce bottlenecks)
  - increased responsiveness
- new protocol for token-based distributed mutual exclusion
- probable owners annotated with priority levels
- set of local queues stores requests
- stored requests age (using local time facilities)
- on token passing, sender's queue merged with receiver's
- merging follows
  - strict priority ordering, then
  - descending age (FIFO within each prio level)
  - same requests in both queue $\Rightarrow$ 1 request w/ max. age
- if head(Q) = self $\Rightarrow$ use token (mutex) else fwd to head(Q)
- no global logical clocks, no timestamps, no broadcast
- $\Rightarrow$ new protocol, same performance as best unprio. prot.
- priority model can also be used to guide thread distribution (more important threads placed on nodes with better performance)

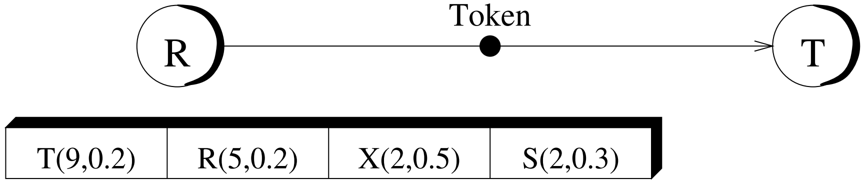
Queue Merging



1) forward, merge, use



2a) before forward to R and merging queues



2b) afterwards, cannot use in R but must forward to T

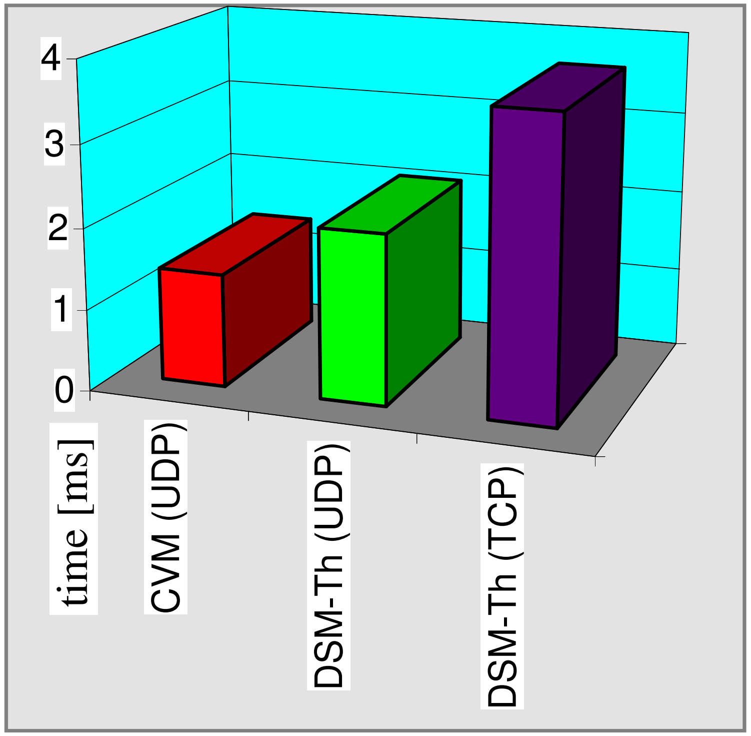
Implementation

- under SunOS 4.1.x and Linux (heterogeneous)
- uses FSU Pthreads library
- page-based SRSW, MRSW; EC coming
- thread distribution implicit (runtime) or explicit (user)
- communication over TCP sockets
- page fault overhead on SPARC classic: 54 ms
- tuning options:
  - communication: stream-lining, UDP, medium
  - runtime: processor frequency
- interpolation: overhead comparable to CVM

- under SunOS 4.1.x and Linux (heterogeneous)
- uses FSU Pthreads library
- page-based SRSW, MRSW; EC coming
- thread distribution implicit (runtime) or explicit (user)
- communication over TCP sockets
- tuning options:
  - within TCP: factor 1.5 (comm.)
  - TCP $\rightarrow$ UDP: factor 3.5 (comm.)
  - 50 MHz $\rightarrow$ 275 MHz: factor 5.5 (runtime)
  - 10 Mbps Ethernet $\rightarrow$ 155 Mbps ATM: factor 15.5 (comm.)
- interpolation: overhead comparable to CVM
  - 1.388 ms for CVM at best (275 Mhz Alpha, UDP, 155 Mbps network)
  - 2.08 ms for DSM-Threads (UDP)
  - 3.65 ms for DSM-Threads (TCP)

Preliminary Results

Description	Time
Entire Page Exchange	54 ms
Page Fault and Signal Handling	2 ms
Invalidation	4 ms
Validation	2 ms
2 TCP Messages	46 ms

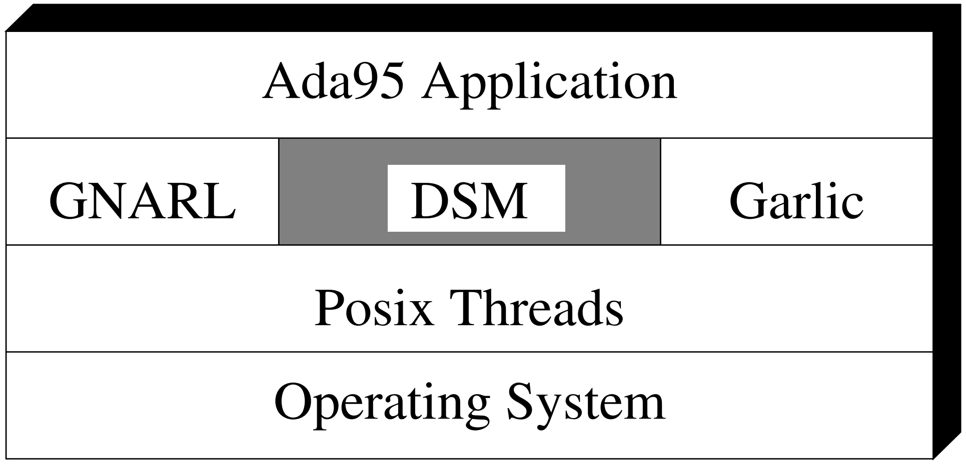


Ada 95 Passive Partitions over DSM

- passive partitions contain data, only
- DSM-Threads integrates with Gnu Ada Translator (Gnat)
- regular data $\Rightarrow$ MRSW
- protected objects $\Rightarrow$ MRMW (EC + new prio protocol)
- support for DSM and distr. synchronization
- Glade supports active partitions (distr. threads)

- passive partitions contain data, only (plain + protected objects)
- DSM-Threads integrates with Gnu Ada Translator (Gnat)
- regular data ⇒ MRSW
  - no modifications for passive partitions (PPs) with plain data
- protected objects ⇒ MRMW (EC + new prio protocol)
  - modify code generator to support POs in PPs
    - protected procedures: distributed write lock
    - protected functions: distributed read lock
  - semantic checks for PPs missing in Gnat
- support for DSM and distr. synchronization
- Glade supports active partitions (distr. threads)
  - Gnatdist: tool for partition configuration
  - enhancements to Gnatdist for initial data placements
  - Garlic: runtime for active partitions
  - centralized server approach (bottleneck)
  - RPC (Ada 95 remote call interface)
  - distributed termination
  - Garlic may be replaced by DSM-Threads

Ada 95 Runtime-System Components



System Dependencies

- POSIX: Threads, `mprotect`, `SIGSEGV`
- fault address known in handler
- Pthreads mutex and cond. var. ops signal-safe
- page alignment for data (Gnu gcc, as, ld)
- DSM data initialized (relaxed by ELF)
- heterogeneous: XDR, DWARF, Gnu BFD

- open problems
  - `mmap` customization (page validation race for MT)
- future work
  - DSM heap allocation

Related Work

- decentralized dynamic manager algorithm
- TreadMarks: lazy release consistency
- Midway: entry consistency
- Filaments: worker threads, dynamic data placement
- Chant: msg passing, RSRs, remote thread ops
- Nexus: RSRs and GPs
- Ariadne: thread migration
- Millipede: multi-threaded nodes, page-based DSM over NT

- Li/Hudak: decentralized dynamic manager algorithm ⇒ MRSW
- extension for token-based distributed mutual exclusion (Naimi et. al.)
- Cox/Keleher's TreadMarks: lazy release consistency, lock page
- Bershad's Midway: entry consistency (but broadcast lock algorithm)
- Lowenthal/Andrews' Filaments:
  - worker threads
  - dynamic data placement (requires compiler modifications)
- Haines' Chant: msg passing, RSRs, remote thread ops
- Foster's Nexus: RSRs and GPs
- page fault is threaded RSR (can't be non-threaded since signal-unsafe)
- Mascarenhas' Ariadne: thread migration (only homogeneously)
- Schuster's Millipede: closest to us, multi-threaded nodes, page-based DSM but over NT instead of POSIX

Summary

- design / implementation DSM threads
  - consistency protocols (SRSW, MRSW, entry)
  - asynchronous access model
  - decentralized locking, synchronization
  - priority enhancements
  - heterogeneous support
- concurrency, responsiveness, tuning
- 15% runtime overhead, rest communication
- interpolation seems to indicate that approach competitive
- best of both worlds (SMP and distributed)
- portable
- base for HLL runtime
- first distributed POSIX-like threads