Distributed Shared-Memory Threads:

DSM-Threads

Frank Mueller

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

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

Overview

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

Motivation

- problem w/ shared-memory multiprocessors (SMPs):
  - bus is bottleneck
- problem w/ distributed programming:
  - 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
- interface close to Pthreads
- minimal modifications to source programs
- options for additional interfaces
- least possible communication overhead
- heterogeneous support
- independent subsystems
- extensible interface
- decentralized approaches

- portability: no modifications to compiler, OS, HW
- interface close to Pthreads (syntactically and semantically)
- minimal modifications to source programs when migrating from Pthreads to DSM-Threads (toward "backwards compatibility")
- options for additional interfaces to fine-tune applications (for expert users)
- least possible communication overhead (since communication may be slow)
- heterogeneous support
- independent subsystems (e.g. communication, simple interface, advanced interface)
- extensible interface (for future enhancements)
- decentralized approaches (to avoid bottlenecks)

Design Issues

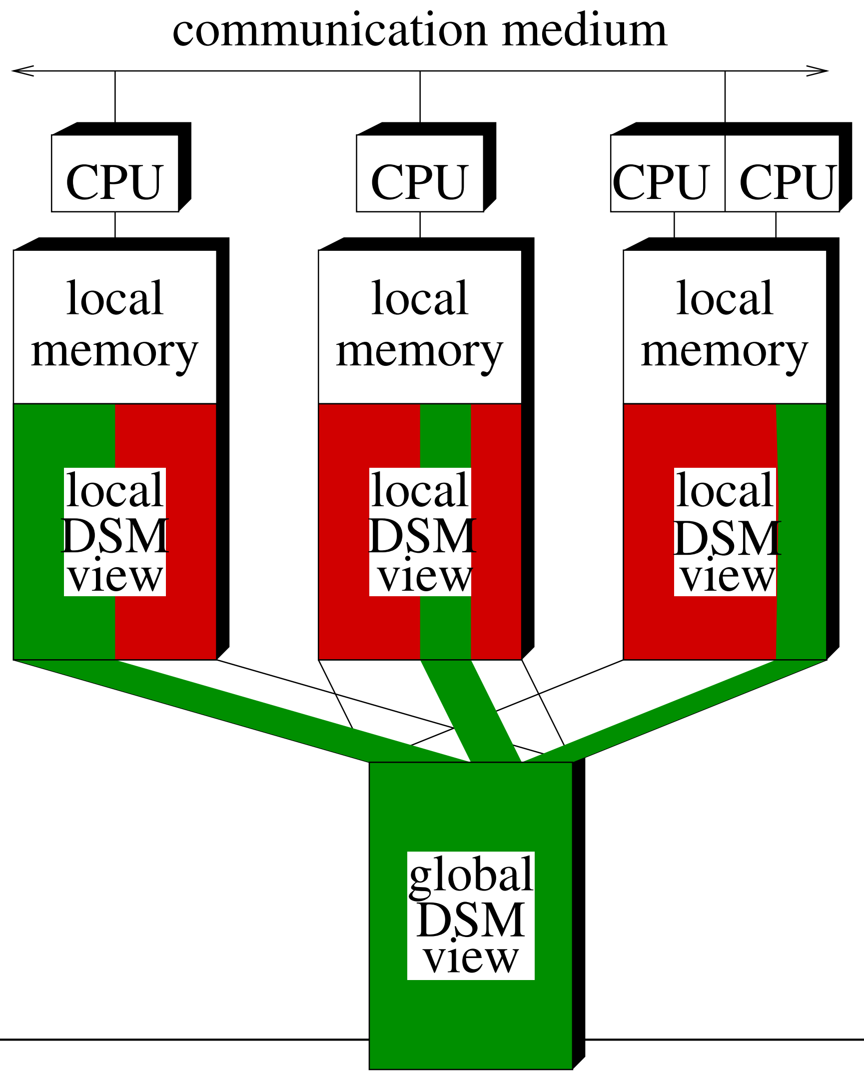
- DSM data identification
- communication
- data consistency and access protocols
- distributed synchronization methods
- data distribution
- thread distribution

Task Model

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

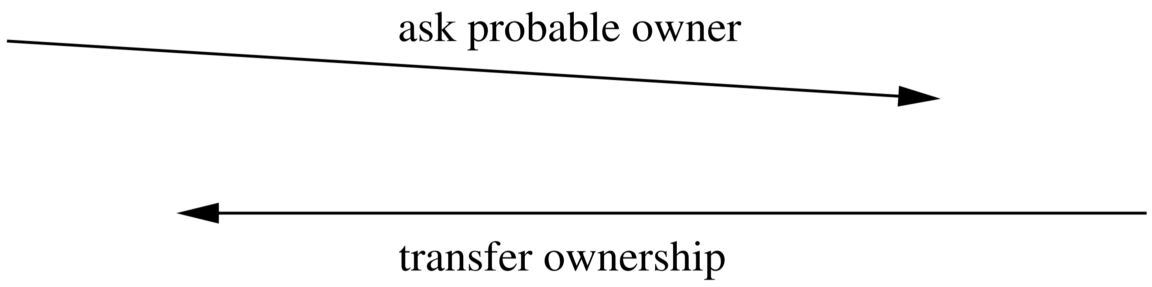
- node = (host, process)
- process: DSM user / system threads, Pthreads
- communication server
  - highest prio
  - open comm. channel, wait for msgs
  - decode msg header
  - act on msgs or delegate
- worker threads
  - process msg
  - handle page faults
  - serve as new user thread
- responsiveness
  - $|$ 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

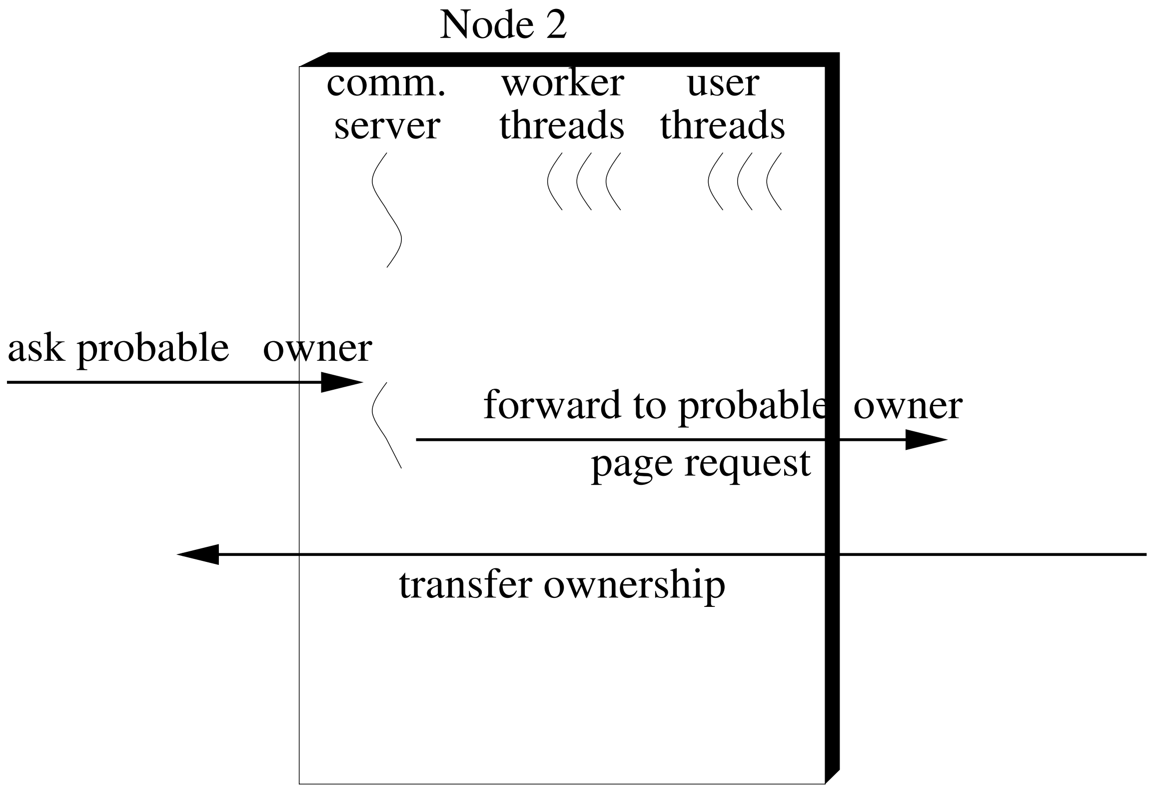

communication medium
CPU
CPU
CPU
CPU
local memory
local memory
local memory
local DSM view
local DSM view
local DSM view
global DSM view

Page Fault Model

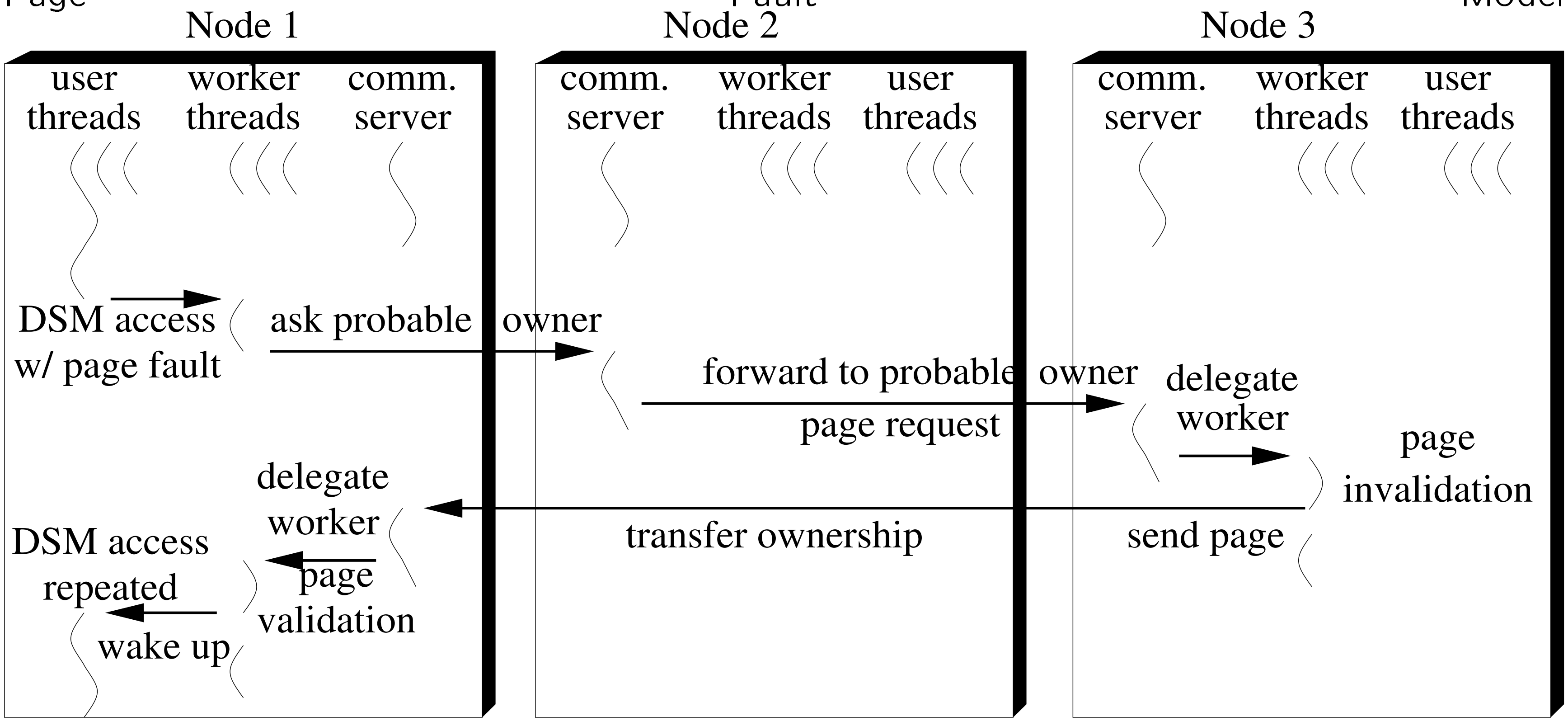
Page Fault Model



Page Fault Model



Page Fault Model



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)

Sample Program

```
#include <dsm.h>

DSM_DATA_BEGIN
int z = 1;     /* DSM data, shared by all nodes       */
float f = 1.0;
DSM_DATA_END

int y = 2;     /* global data, separate for each node */

void new_thread(void *arg)
{
...
}

void main(int argc, char **argv)
{
  dsm_attr_t attr; /* local data */
  dsm_thread_t dsm_thread;
...
  dsm_init(argc, argv);
...
  dsm_attr_init(&attr);
  dsm_attr_setnode(&attr, node); /* node DSM-thread runs on */
  dsm_thread_create(&dsm_thread, &attr, new_thread, arg);
...
  dsm_exit(NULL);
}
```

Communication Protocols

- message passing protocol
- one implementation w/ sockets (reliable, no loss)
- can be replaced by implementations
- 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

Synchronization

- mutual exclusion (locks)
  - new protocol for prio token-based distributed mutex
  - optional: read and write locks distinguished
  - piggy-back data for entry consistency page faults
- conditional synchronization (condition variables)
  - signal owner of locked mutex
- barriers
  - tree barrier
  - thread groups form barrier by registration
  - bottom-up synchronization
  - imbalance when de-registered

- mutual exclusion (locks)
  - new protocol for prio token-based distributed mutex
  - w/o prios same as Li/Hudak or Naimi et. al.
  - optional: read and write locks distinguished
  - piggy-back data for entry consistency (MRMW) ⇒ no page faults
  - local mutex owner table
- conditional synchronization (condition variables)
  - signal owner of locked mutex
  - one mutex per cond. var., search for owner in mutex table
- barriers
  - tree barrier
  - thread groups form barrier by registration
  - 1st thread is root
  - other threads alternate transitively right/left
  - bottom-up synchronization
  - imbalance when de-registered (fixed by more registrations)

Data Distribution

- collect trace data
- reorder DSM data layout
- preprocessor support
- requires recompilation
- OO / compiler mods would allow changes to indirections

Thread Distribution

- explicit by programmer
- implicit if not specified
- trace-based for repeated executions

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 / Future Work

- `mmap` customization (page validation race for MT)
- DSM heap allocation
- priority handling
- performance evaluation
- Ada95 passive partitions

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

- Li/Hudak: decentralized dynamic manager algorithm $\Rightarrow$ 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)

Summary

- design / implementation DSM threads
  - consistency protocols (SRSW, MRSW, entry)
  - decentralized locking, synchronization
  - restrictions, problems
- best of both worlds (SMP and distributed)
- portable, heterogeneous
- base for HLL runtime
- first distributed POSIX-like threads