

DISTRIBUTED SYSTEMS (COMP9243)

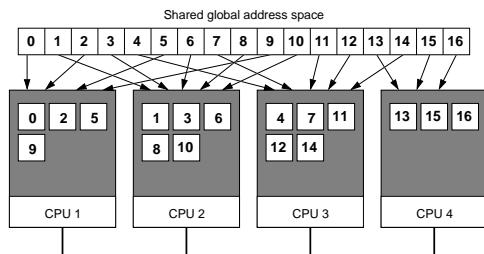
Lecture 3b: Distributed Shared Memory

Slide 1

- ① DSM
- ② Case study
- ③ Design issues
- ④ Implementation issues

DISTRIBUTED SHARED MEMORY (DSM)

DSM: shared memory + multicomputer



Slide 2

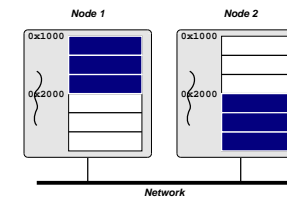
SHARED ADDRESS SPACE

DSM consists of two components:

- ① Shared address space
- ② Replication and consistency of memory objects

Shared address space:

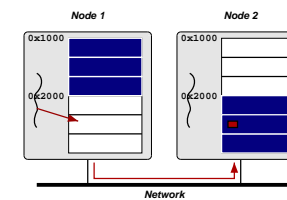
Slide 3



→ Shared addresses are valid in all processes

Transparent remote access:

Slide 4



Properties:

- Remote access is expensive compared to local memory access
- Individual operations can have very low overhead
- Threads can distinguish between local and remote access

Why DSM?:

- Shared memory model: easiest to program to
- Physical shared memory not possible on multicomputer
- DSM emulates shared memory

Benefits of DSM:

Slide 5

- Ease of programming (shared memory model)
 - Eases porting of existing code
 - Pointer handling
 - Shared pointers refer to shared memory
 - Share complex data (lists, etc.)
 - No marshalling
-

DSM ENVIRONMENTS

Slide 6

- Multiprocessor
 - NUMA
 - Multicomputer
 - Supercomputer
 - Cluster
 - Network of Workstations
 - Wide-area
-

APPLICATIONS OF DSM

Slide 7

- Scientific parallel computing
 - Bioinformatics (gene sequence analysis)
 - Simulations (climate modeling, economic modeling)
 - Data processing (physics, astronomy)
 - Graphics (image processing, rendering)
 - Data server (distributed FS, Web server)
 - Data storage
-

CHALLENGES OF DSM

Slide 8

- Transparency:
- Location, migration, replication, concurrency
- Reliability:
- Computations depend on availability of data
- Performance:
- Important in high-performance computing
 - Important for transparency
- Scalability:
- Important in wide-area
 - Important for large computations
-

Slide 9

Consistency:

- Access to DSM should be consistent
- According to a consistency model

Programmability:

- Easy to program
 - Communication transparency
-

Middleware:

→ Library:

- Library routines to create/access shared memory
- Example: MPI-2, CRL

Slide 11

→ Language

- Shared memory encapsulated in language constructs
 - Extend language with annotations
 - Example: Orca, Linda, JavaSpaces, JavaParty, Jackal
-

DSM IMPLEMENTATIONS

Hardware:

- Multiprocessor
- Example: MIT Alewife, DASH

OS with hardware support:

Slide 10

- SCI network cards (SCI = Scalable Coherent Interconnect)
- SCI maps extended physical address space to remote nodes
- OS maps shared virtual address space to SCI range

OS and Virtual Memory:

- Virtual memory (page faults, paging)
 - Local address space vs Large address space
-

Typical Implementation:

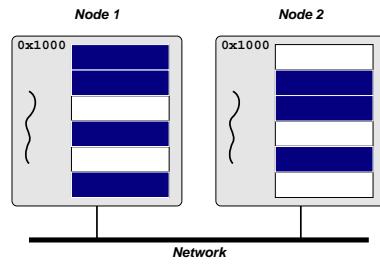
- Provided by some research OSes (e.g., Mach and Chorus)
- Most often implemented in user space (e.g., TreadMarks, CVM)
- User space: what's needed from the kernel?

Slide 12

- User-level fault handler
(e.g., Unix signals)
 - User-level VM page mapping and protection
(e.g., `mmap()` and `mprotect()`)
 - Message passing layer
(e.g., socket API)
-

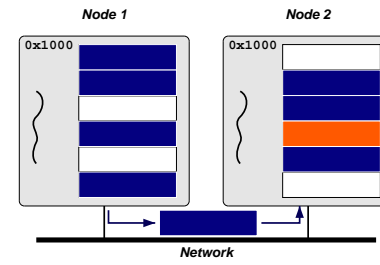
Example: two processes sharing memory pages:

Slide 13



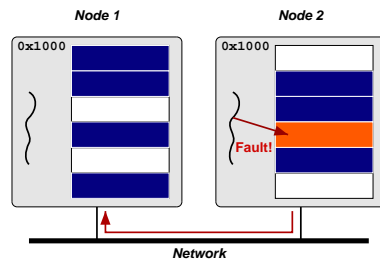
Page migration and replication:

Slide 15



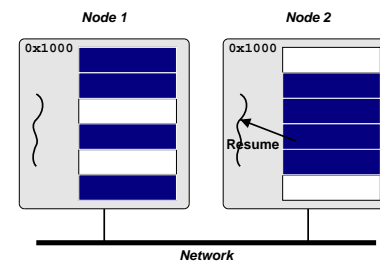
Occurrence of a read fault:

Slide 14



Recovery from read fault:

Slide 16



DSM MODELS

Shared page (coarse-grained):

- Traditional model
- Ideal page size?
- ✗ False sharing
- Examples: Ivy, TreadMarks

Slide 17

Shared region (fine-grained):

- More fine grained than sharing pages
- ✓ Prevent false sharing
- ✗ Not regular memory access (transparency)
- Examples: CRL (C Region Library), MPI-2 one-sided communication, Shasta

Example: Iterative solver using MPI-2 one-sided communication:

```
while(!converged(A)){
  update(A);
  MPI_Win_fence(MPI_MODE_NOPRECEDE, win);
  for(i=0; i < toneighbors; i++)
    MPI_Put(&frombuf[i], 1, fromtype[i], toneighbor[i],
           todisp[i], 1, totype[i], win);
  MPI_Win_fence((MPI_MODE_NOSTORE | MPI_MODE_NOSUCCEED), win);
}
```

Slide 18

- Note how multiple MPI_Put() are issued per fence call
 - Barriers partition the program into phases
 - Communication and computation may overlap within a phase
-

Shared variable:

- Release and Entry based consistency
- Annotations
- ✓ Fine grained
- ✗ More complex for programmer
- Examples: Munin, Midway

Slide 19

Shared structure:

- Encapsulate shared data
 - Access only through predefined procedures (e.g., methods)
 - ✓ Tightly integrated synchronisation
 - ✓ Encapsulate (hide) consistency model
 - ✗ Lose familiar shared memory model
 - Examples: Orca (shared object), Linda (tuple space)
-

LINDA EXAMPLE

```
main() {
  ...
  eval("function", f());
  eval("function", f());
  ...
  for (i=0; i<100; i++)
    out("data", i);
  ...
}
f(){
  ...
  in("data", ?x);
  y = g(x);
  out("function", x, y);
  ...
}
```

Slide 20

CASE STUDY

TreadMarks:

- 1992 Rice University
- Page based DSM library
- C, C++, Java, Fortran
- Lazy release consistency model
- Heterogeneous environment

Slide 21

USING DSM

Initialisation:

- Compiling
- Starting program
- Starting services
- Specifying nodes
- Finding nodes

Processes:

- Start
- Stop
- Process Ids

Slide 22

Shared Memory:

- Allocating
- Accessing
- Freeing

Synchronisation:

- Locks
- Barriers

USING TREADMARKS

Compiling:

- Compile
- Link with TreadMarks libraries

Starting a TreadMarks Application:

```
app -- -h host1 -h host2 -h host3 -h host4
```

Slide 23

Anatomy of a TreadMarks Program:

- Starting remote processes

```
Tmk_startup(argc, argv);
```

- Allocating and sharing memory

```
shared = (struct shared*) Tmk_Malloc(sizeof(shared));  
Tmk_distribute(&shared, sizeof(shared));
```

- Barriers

```
Tmk_barrier(0);
```

Slide 24

- Acquire/Release

```
Tmk_lock_acquire(0);  
shared->sum += mySum;  
Tmk_lock_release(0);
```

TREADMARKS APPLICATION EXAMPLE

Slide 25

full code example

DESIGN ISSUES

Slide 26

- Structure and Granularity
- Replication
- Synchronisation primitives
- Heterogeneity
- Scalability
- Fault tolerance
- Security

Granularity

- Page based
- Page size: minimum system page size

Replication

- Lazy release consistency

Scalability

- Meant for cluster or NOW

Slide 27

Synchronisation primitives

- Locks (acquire and release), Barrier

Heterogeneity

- Limited (doesn't address endianness or mismatched word sizes)

Fault Tolerance

- Research

No Security

IMPLEMENTATION ISSUES

Slide 28

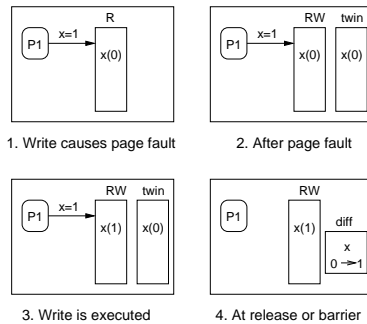
- Consistency protocol
- Update propagation
- Communication
- Data location
- Modification detection
- Memory management
- Thrashing
- Heterogeneity
- Implementation of synchronisation primitives
- Implementation of fault tolerance
- Implementation of security

TREADMARKS IMPLEMENTATION

Consistency Protocol:

- Multiple writer
- Twins
- Reduce false sharing

Slide 29



Update Propagation:

- Modified pages invalidated at acquire
- Page is updated at access time
- Updates are transferred as diffs

Lazy Diffs:

- Slide 30
- Normally make diffs at release time
 - Lazy: make diffs only when they are requested

Communication:

- UDP/IP or AAL3/4 (ATM)
- Light-weight, user-level protocols to ensure message delivery
- Use SIGIO for message receive notification

Data Location:

- Know who has diffs because of invalidations
- Each page has a statically assigned manager

Modification Detection:

- Slide 31
- Page Fault
 - If page is read-only then do consistency protocol
 - If not in local memory, get from manager

Memory Management:

- Garbage collection of diffs

Initialisation:

- Slide 32
- Processes set up communication channels between themselves
 - Register SIGIO handler for communication
 - Allocate large block of memory
 - Same (virtual) address on each machine
 - Mark as non-accessible
 - Assign manager process for each page, lock, barrier (round robin)
 - Register SEGV handler

READING LIST

Distributed Shared Memory: A Survey of Issues and Algorithms

An overview of DSM and key issues as well as older DSM implementations.

Slide 33

TreadMarks: Shared Memory Computing on Networks of Workstations

An overview of TreadMarks, design decisions and implementation.
