Distributed Shared Memory

Abhay Kasturia, Nakul Camasamudram, Rashmi Dwaraka

CS5600 Computer Systems

Computer Systems
Prof. Kapil Arya
Northeastern University
December 2016

Abstract

This project provides an overview of Distributed Shared Memory system implementation at the operating system level. In addition, it is integrated with memory region invalidation based DSM implementation to provide consistency across the processes accessing the memory distributed across various systems. This DSM implementation provide the user with a transparent, efficient and scalable shared memory programming environment.

As part of the project, we implemented a library which executes in Linux user space, which is a Distributed Shared Memory system similar to Ivy. This is achieved by instantiating a Centralized Memory Management Server which synchronizes the shared pages across the nodes on a network and uses a write invalidation protocol to synchronize the shared content across clients.

The library is initialized in each client which establishes the socket connection with the centralized manager. Then it helps synchronize the shared pages using the calls defined in RPC. The centralized manager on the other hand maintains a list of shared pages along with the list of machine nodes accessing those shared page.

The client applications can then be initialized in parallel distributed across the connected machine nodes, where each client program is aware of their share of work. This in our test programs is achieved by using arguments to the client program. The DSM implementation provides performance improvement for application which requires little more space to execute efficiently in less time.

Contributions

(a) **Design and Concepts Research work:**
Abhay , Nakul , Rashmi

(b) **Shared Memory Library implementation in user space:**
Abhay and Nakul

(c) **Central Manager for Distributed Shared Memory co-ordinations:**
Nakul

(d) **Handling Network connections and Message handlers:**
Rashmi

(e) **Performance testing and Benchmarking:**
Abhay

(f) **Documentation:**
Abhay and Rashmi

Contents

1. Introduction

In a tightly coupled multiprocessor, accessing main memory via a common bus becomes a serialization point - that limits system size to tens of processors. Distributed-memory multiprocessors overcomes this drawback.These systems consist of a collection of independent computers connected by a high-speed interconnection network. All the communication between the parallel executing processes via shared memory abstraction gives the systems an illusion of physically shared memory and allows the applications to fully utilize the shared-memory paradigm.

A Memory management server makes a global physical memory equally accessible to all processors of the distributed setup. The scalable nature of distributed-memory systems enables the systems with very high computing power. However, communication between processes residing on different nodes involves a message-passing model that requires explicit use of send-receive primitives. The manager is responsible of taking care of data distribution across the system and manage the communication. This also allows the process migration smoothly.

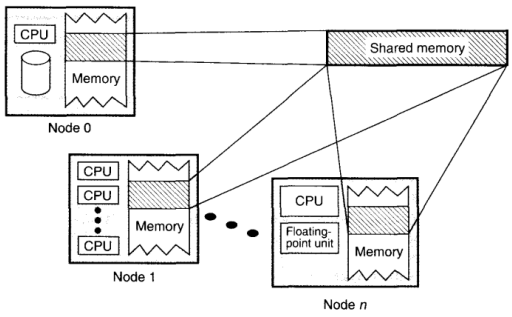

Figure 1.1: Distributed Shared Memory Architecture

The advantages of the Distributed Shared Memory systems are:

- Data sharing is implicit, hiding data movement (as opposed to 'Send'/'Receive' in message passing model)

- Passing data structures containing pointers is easier (in message passing model data moves between different address spaces)
- Moving entire object to user takes advantage of locality difference
- Less expensive to build than tightly coupled multiprocessor system: off-the-shelf hardware, no expensive interface to shared physical memory
- Very large total physical memory for all nodes: Large programs can run more efficiently
- No serial access to common bus for shared physical memory like in multiprocessor systems
- Programs written for shared memory multiprocessors can be run on DSM systems with minimum changes

The advantages of DSM can be realized with a low runtime overhead. There are various approaches that have been suggested to implement the DSM systems:

(1) Hardware implementations that extend traditional techniques to scalable architectures.

(2) Operating system and library implementations that achieve sharing and coherence through virtual memory-management mechanisms

(3) Compiler implementations where shared accesses are automatically converted into synchronization and coherence primitives.

2. Novelty: What is the core contribution that is new

DSM is not implemented in Linux and the library we are building extend DSM capabilities to code that is run on Linux. This makes

3. Design/Approach

Our Design for implementation of Distributed Shared Memory Systems involves three parts:

(a) **Centralized Memory Management Server:**
The centralized manager runs on one of the instances of the distributed system setup. The Manager will create a socket connection for each client. The centralized manager manages the shared regions with Page Table. Each page in the shared memory registered is managed with the help of page table. The page table includes the parameters like
(1) Lock - To protect the page from multiple writes
(2) Client[] - Machine nodes accessing this particular page
(3) Permission - read/write access to the page
(4) Invalidation - True/False
(5) Encoded-page

(b) **Networking and Message Handler:**
This module of the project handles the page request signals, interprets the type of access and is responsible for sending and receiving data between manager and client.

(c) **Library package for user applications:**
The library is pre-compiled along with the client application. The Application user can choose to utilize the shared memory utility by initializing the shared memory and registering with the centralized manager. Once the shared region and the connection to the manager is set.
Application program causes a segmentation fault, operating system sends SIGSEGV signal. The library checks for the signal, if it is not SIGSEGV or if the page doesnot belong to the shared memory region, then the default page handler of the operating system is invoked. If the Page requested belong to the Shared memory region, then the library redirects the request to the manager with appropriate read or write request.

4. Implementation / Technical Challenges

(a) **Failed implementation on xv6**
The initial motivation of the project was to implement the distributed shared memory on xv6 operating system. xv6 is a very basic operating system which is built for teaching purpose by MIT. As a basic Operation system, xv6 lacks a network stack to connect to other systems, one of the crucial section of the project. Given the time constraints, we were not able to implement the network stack on top xv6 to form a mesh network of xv6 systems. So, we went ahead and tried to use some off the shelf network stack on top of xv6. We were not able to achieve this as well, due to lack of proper documentation for the existing implementation available.

Due to time constraints, we went ahead with the main implementation of the distributed shared memory concept on to much more stable and established operating system like Linux. Due to the complexity of the Linux operating system, we continued to implement the DSM architecture at user space. As part of the future scope, we would like to push the implementation to the Linux kernel space. But the current implementation can be used as a plugin and use the advantages of the shared memory across systems.

(b) **Implementation Challenges**
It was a great learning curve as none of us were experienced in linux system programming. We had to debug lot of segmentation faults while implementing the memory protocols.

5. Evaluation

Matrix operations find various applications in highly computationally intensive task such as high dimensional problems in statistical physics, graph theory, searching and sorting. Therefore, such operations are often used as a benchmark to measure a supercomputer's computing capabilities. While Matrix operations are not the only operations employed by a distributed system, because of its wide spread usage and importance, we choose Matrix Multiplication program as a benchmark to demonstrate the advantages of shared memory architecture.

Table 5.1: Performance Statistics

Approach	Number of Nodes	Matrix-Size	Time(ms)
1	1	100 × 100	0
	1	500 × 500	0
	1	1000 × 1000	0
2	3	100 × 100	0
	3	500 × 500	0
	3	1000 × 1000	0
3	5	100 × 100	0
	5	500 × 500	0
	5	1000 × 1000	0
4	10	100 × 100	0
	10	500 × 500	0
	10	1000 × 1000	0

6. Conclusion and Future Work

(1) DSM Kernel level implementation

(2) Implement fault tolerance guarantees

(3) Checkpointing

(4) Process migration across the machines in the network

(5) Implement Distributed manager, which would improve the efficiency for large scale Nodes

A. Appendix

The code implementation can be found in the below mentioned link:
https://github.ccs.neu.edu/dwarakarashmi/cs5600_fl16_DSM_project

The presentation used for the demo:
https://docs.google.com/presentation/d/1IN34QxhEPbS1-XLLYihGAEv3lh_I13s3Drr_1MNmh10/edit?ts=584e820f#slide=id.g1a0c2d09f5_0_60

The demo video can be found below:
youtube.com/updatethis

B. References

(a) http://www.cdf.toronto.edu/~csc469h/fall/handouts/nitzberg91.pdf

(b) https://en.wikipedia.org/wiki/Distributed_shared_memory

(c) https://github.com/mit-pdos/xv6-public

(d) http://www.alexonlinux.com/signal-handling-in-linux

(e) http://stackoverflow.com

(f) https://docs.python.org/2/howto/sockets.html

(g) http://www.linuxhowtos.org/C_C++/socket.htm