CHAPTER 10

DISTRIBUTED SHARED MEMORY

10.1 INTRODUCTION

Traditionally, distributed computing has been based on the message passing model in which processes interact and share data with each other by exchanging data in the form of messages. Hoare's communicating sequential processes (Sec. 2.6.4), the client-server model (Sec. 4.5.9), and remote procedure calls (Sec. 4.7.2) are examples of this model.

Distributed shared memory (DSM) system is a resource management component of a distributed operating system that implements the shared memory model in distributed systems, which have no physically shared memory (Fig. 10.1). The shared memory model provides a virtual address space that is shared among all nodes (computers) in a distributed system.

10.2 ARCHITECTURE AND MOTIVATION

With DSM, programs access data in the shared address space just as they access data in traditional virtual memory. In systems that support DSM, data moves between secondary memory and main memory as well as between main memories of different nodes. Each node can own† data stored in the shared address space, and the ownership can change

†Typically, the node which creates a data object owns the data object initially.

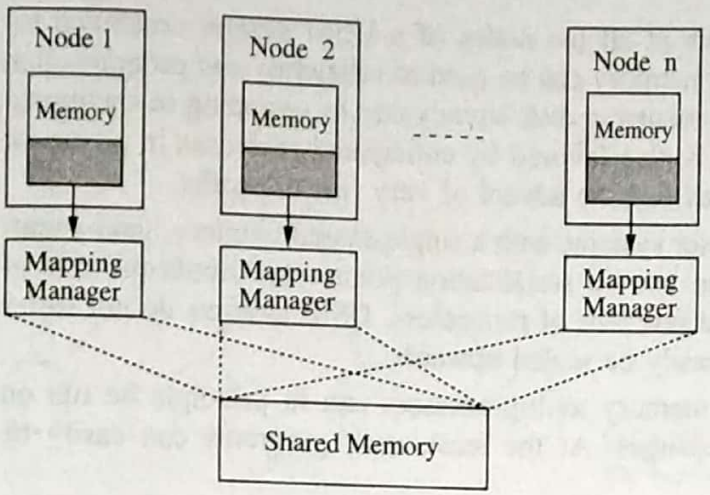

FIGURE 10.1
Distributed shared memory (adapted from [28]).

when data moves from one node to another. When a process accesses data in the shared address space, a *mapping manager* maps the shared memory address to the physical memory (which can be local or remote). The mapping manager is a layer of software implemented either in the operating system kernel or as a runtime library routine. To reduce delays due to communication latency, DSM may move data at the shared memory address from a remote node to the node that is accessing data (when the shared memory address maps to a physical memory location on a remote node). In such cases, DSM makes use of the communication services of the underlying communication system.

Advantages of Distributed Shared Memory are:

1. In the message passing model, programs make shared data available through explicit message passing. In other words, programmers need to be conscious of the data movement between processes. Programmers have to explicitly use communication primitives (such as SEND and RECEIVE), a task that places a significant burden on them. In contrast, DSM systems hide this explicit data movement and provide a simpler abstraction for sharing data that programmers are already well versed with. Hence, it is easier to design and write parallel algorithms using DSM rather than through explicit message passing.
2. In the message passing model, data moves between two different address spaces. This makes it difficult to pass complex data structures between two processes. Moreover, passing data by reference and passing data structures containing pointers is generally difficult and expensive. In contrast, DSM systems allow complex structures to be passed by reference, thus simplifying the development of algorithms for distributed applications.
3. By moving the entire block or page containing the data referenced to the site of reference instead of moving only the specific piece of data referenced, DSM takes advantage of the locality of reference exhibited by programs and thereby cuts down on the overhead of communicating over the network.
4. DSM systems are cheaper to build than tightly coupled multiprocessor systems. This is because DSM systems can be built using off-the-shelf hardware and do not require complex interfaces to connect the shared memory to the processors.

5. The physical memory available at all the nodes of a DSM system combined together is enormous. This large memory can be used to efficiently run programs that require large memory without incurring disk latency due to swapping in traditional distributed systems. This fact is also favored by anticipated increases in processor speed relative to memory speed and the advent of very fast networks.
6. In tightly coupled multiprocessor systems with a single shared memory, main memory is accessed via a common bus—a serialization point—that limits the size of the multiprocessor system to a few tens of processors. DSM systems do not suffer from this drawback and can easily be scaled upwards.
7. Programs written for shared memory multiprocessors can in principle be run on DSM systems without any changes. At the least, such programs can easily be ported to DSM systems.

In essence, DSM systems strive to overcome the architectural limitations of shared memory machines and to reduce the effort required to write parallel programs in distributed systems.

10.3 ALGORITHMS FOR IMPLEMENTING DSM

The central issues in the implementation of DSM are: (a) how to keep track of the location of remote data, (b) how to overcome the communication delays and high overhead associated with the execution of communication protocols in distributed systems when accessing remote data, and (c) how to make shared data concurrently accessible at several nodes in order to improve system performance. We now describe four basic algorithms to implement DSM systems [31].

10.3.1 The Central-Server Algorithm

In the central-server algorithm [31], a central-server maintains all the shared data. It services the read requests from other nodes or clients by returning the data items to them (see Fig. 10.2). It updates the data on write requests by clients and returns acknowledgment messages. A timeout can be employed to resend the requests in case of failed acknowledgments. Duplicate write requests can be detected by associating sequence numbers with write requests. A failure condition is returned to the application trying to access shared data after several retransmissions without a response.

While the central-server algorithm is simple to implement, the central-server can become a bottleneck. To overcome this problem, shared data can be distributed among several servers. In such a case, clients must be able to locate the appropriate server for every data access. Multicasting data access requests is undesirable as it does not reduce the load at the servers compared to the central-server scheme. A better way to distribute data is to partition the shared data by address and use a mapping function to locate the appropriate server.

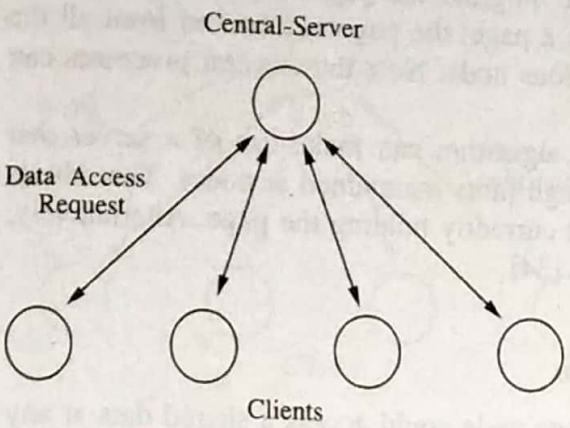

FIGURE 10.2
The central-server algorithm (adapted from [31]).

10.3.2 The Migration Algorithm

In contrast to the central-server algorithm, where every data access request is forwarded to the location of data, data in the migration algorithm is shipped to the location of the data access request, allowing subsequent accesses to the data to be performed locally [31] (see Fig. 10.3). The migration algorithm allows only one node to access a shared data at time.

Typically, the whole page or block containing the data item migrates instead of an individual item requested. This algorithm takes advantage of the locality of reference exhibited by programs by amortizing the cost of migration over multiple accesses to the migrated data. However, this approach is susceptible to *thrashing*, where pages frequently migrate between nodes while servicing only a few requests.

To reduce thrashing, the Mirage system [18] uses a tunable parameter that determines the duration for which a node can possess a shared data item. This allows a node to make a number of accesses to the page before it is migrated to another node. The Munin system [5] strives to reduce data movement by employing protocols that are appropriate to different data access patterns (see Sec. 10.5.3 for details).

The migration algorithm provides an opportunity to integrate DSM with the virtual memory provided by the operating system running at individual nodes. When the page size used by DSM is a multiple of the virtual memory page size, a locally held shared memory page can be mapped to an application's virtual address space and accessed using normal machine instructions. On a memory access fault, if the memory address

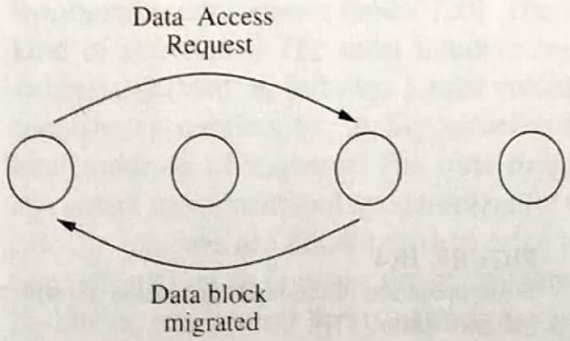

FIGURE 10.3
The migration algorithm (adapted from [31]).

maps to a remote page, a fault-handler will migrate the page before mapping it to the process's address space. Upon migrating a page, the page is removed from all the address spaces it was mapped to at the previous node. Note that several processes can share a page at a node.

To locate a data block, the migration algorithm can make use of a server that keeps track of the location of pages, or through hints maintained at nodes. These hints direct the search for a page toward the node currently holding the page. Alternatively, a query can be broadcasted to locate a page [34].

10.3.3 The Read-Replication Algorithm

In previous approaches, only processes on one node could access a shared data at any one moment. The read-replication algorithm [31] extends the migration algorithm by replicating data blocks and allowing multiple nodes to have read access or one node to have read-write access (the multiple readers-one writer protocol). Read-replication can improve system performance by allowing multiple nodes to access data concurrently. However, the write operation is expensive as all the copies of a shared block at various nodes will either have to be invalidated (see Fig. 10.4) or updated with the current value to maintain the consistency of the shared data block.

In the read-replication algorithm, DSM must keep track of the location of all the copies of data blocks. In the IVY system [27], the owner node of a data block keeps track of all the nodes that have a copy of the data block. In the PLUS system [8], a distributed linked-list is used to keep track of all the nodes that have a copy of the data block.

Nevertheless, read-replication has the potential to reduce the average cost of read operations when the ratio of reads to writes is large. Many read-replication algorithms implemented in the IVY system are described in Sec. 10.7.1.

10.3.4 The Full-Replication Algorithm

The full-replication algorithm [31] is an extension of the read-replication algorithm. It allows multiple nodes to have both read and write access to shared data blocks (the multiple readers-multiple writers protocol). Because many nodes can write shared data concurrently, the access to shared data must be controlled to maintain its consistency.

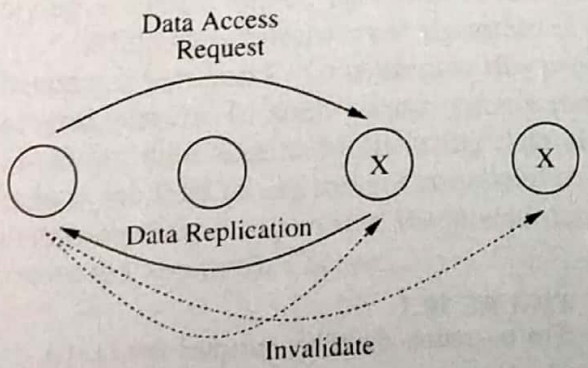

FIGURE 10.4
Write operation in the read-replication algorithm (adapted from [31]).