Chapter One

Introduction

Distributed Shared Memory (DSM) Multicomputers have the potential to provide high-performance by exploiting parallelism, while allowing the programmer to express the parallelism using the familiar shared memory programming model. The multicomputer type of system [Atha88] is composed of a large number of independent processing nodes (comprising processor as well as local memory) which are interconnected via point-to-point links. Unlike shared-memory multiprocessors [Thak88, Wils87, Good88] where processors are connected by buses, the multicomputer has no single component, such as a common bus or shared memory, which can become a performance bottleneck and whose correct operation is critical for the operation of the system. Multicomputers are, hence, capable of scaling to sizes larger than shared memory multiprocessors.

While the memory in a multicomputer is distributed, the DSM multicomputer provides a shared memory programming model where processes running on different nodes can execute in the the same address space. Interprocess communication in this environment is accomplished by reads and writes to shared memory, rather than through message-passing primitives such as send and receive. The shared memory model provides a straightforward means of developing applications that require different processes to share access to a large common data structure (e.g. a memory-resident collection of images), and applications that require complex data structures to be passed by passing pointers. Support for the shared memory programming model in new systems is also needed to facilitate porting of the many existing applications written for this model. Another advantage of the shared memory model is that process migration (e.g., for load balancing) becomes relatively cheap — in order to migrate a process only the contents of a few registers need to be migrated while the memory can remain in place.

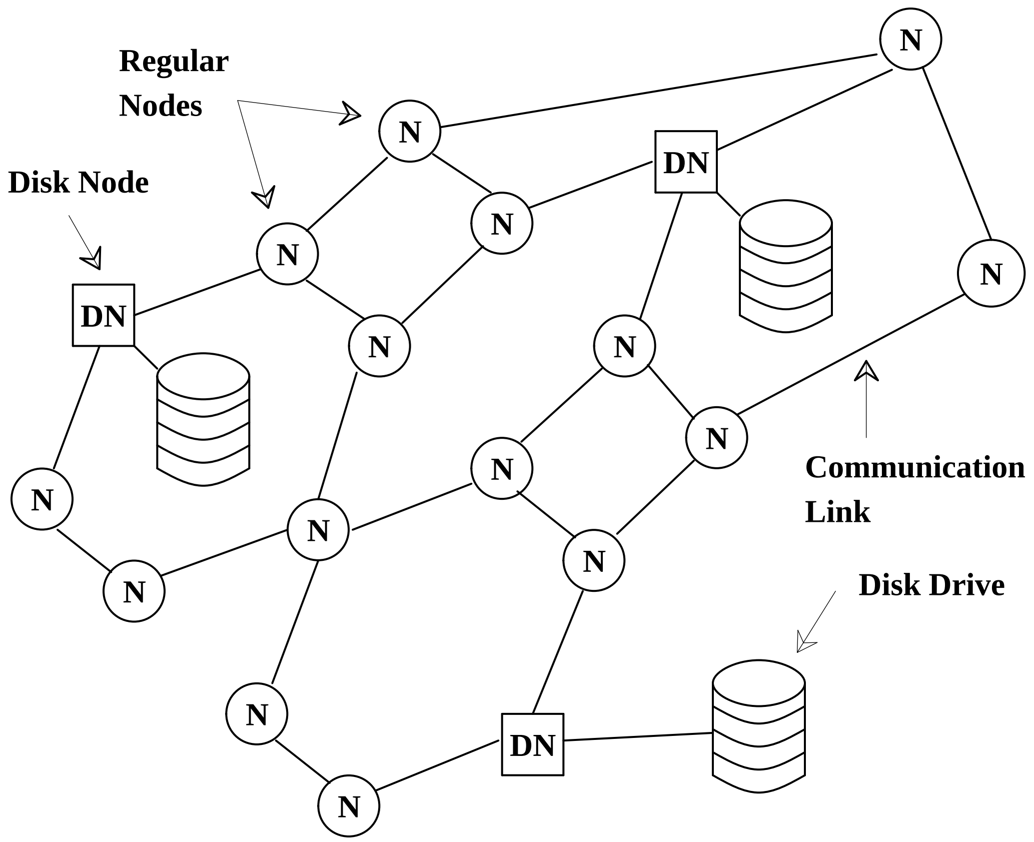

Figure 1.1: Multicomputer.

DSM multicomputers allow individual nodes to cache the data from the distributed shared address space locally to support better performance by reducing the number of remote accesses. If the word requested by a process is not available locally, the block containing the word is read either from a remote memory or from a disk and cached in local memory. The block resides in local memory until another processor requests exclusive access to it, or, as in conventional virtual memory, until it is replaced by another block. Copies of any block may exist in multiple nodes. The DSM implementation employs a coherence protocol to maintain the node memories coherent, and guarantees that process execution is consistent.

DSM multicomputers have been the focus of significant research and development [Li86b, Cher91, Hage92, Burk92, Leno90, Kusk94, Agar95] due to its advantages. These efforts have investigated various issues in caching policies, coherence protocols, consistency models, hardware organization etc. Many of the proposed or implemented systems have not fully addressed critical memory mapping and memory management issues that arise while supporting DSM on a large scale multicomputer. We address these issues in this dissertation and propose a solution based on hierarchical management [Tami92] that manages the distributed shared memory more efficiently than conventional single level schemes, making it suitable for large systems.

The large number of components in a DSM multicomputer increases the likelihood that some component of the system may fail at some time. Since the shared address space is distributed and replicated, such failures have the potential to halt progress of the entire system. Fault-tolerance support is needed to tolerate such component failures and improve the reliability of the system. Fault-tolerance schemes that have been proposed for DSM systems by other researchers [Wu90a, Tam90, Stum90] have adopted approaches which incur high overhead during normal operation and are more expensive for communication-intensive applications. We propose an application-transparent recovery scheme based on coordinated checkpoint and rollback for the DSM multicomputer [Jana94] that can potentially offer better performance than previous recovery schemes.

1.1. Hierarchically Managed DSM

Most existing techniques for providing distributed shared memory on multicomputers perform the mapping, data transfers, and ownership management at a single granularity of blocks (or pages). If large block sizes (pages) are used [Li89a], these techniques suffer from unnecessary long latencies for remote requests, they require higher network bandwidth than is inherently needed by the application, and they lead to unnecessary transfers due to false sharing. A smaller block size (blocks) [Leno90, Kusk94, Agar95] will lead to increased block ownership time [Bayl89], a decrease in miss ratio [Dubo88], and is expected to reduce remote access latencies and the required network bandwidth. A straightforward implementation at a block level for a large DSM system, however, will result in unacceptably high storage overhead for the various mapping tables.

We present *hierarchical management* [Tami92] as a solution to this problem. The proposed scheme dynamically distinguishes between shared and private pages in order to manage each class efficiently. Pages that are not shared are mapped and managed only at the granularity of pages, reducing mapping overhead. Shared pages, however, are managed at both the page and block level, to support efficient sharing. This distributes the coherency load according to the characteristics of the application. Our evaluation indicates that significant performance improvement over conventional page-level schemes are possible with our two-level scheme, without incurring the storage overhead that makes uniform block-level schemes impractical in large systems.

1.2. Reliable DSM

Most error recovery schemes are based on periodically saving a correct state (checkpointing) in reliable storage, and restoring a previously saved state (rollback) upon an error in the current state. Since process states in a DSM system are interdependent due to interprocess communication, when a process rolls back to a checkpointed state at recovery, the states of other processes that communicated with it prior to failure may become invalid. Hence, other processes may also need to be rolled back (propagation of rollback). Error recovery mechanisms proposed for DSM systems (as for message passing systems) must ensure that process states are consistent [Rand78, Chan85] after recovery from an error.

Recovery schemes that have previously been proposed for DSM systems [Wu90a, Tam90, Stum90] all attempt to avoid the cost of rollback propagation by forcing a checkpoint or logging when processes ‘‘communicate.’’ The elimination of rollback propagation make recoveries, which are infrequent, cheap. But, due to the frequent checkpointing or logging, the system load during normal operation is high. The performance degradation is particularly high for communication-intensive applications [Fraz89].

Coordinated checkpointing and rollback based error recovery is an alternative approach used in message-passing systems that incurs lower overhead during normal operation of the system [Bari83, Tami84, Koo87, Tami89]. In this approach, process interactions are tracked, and processes that have communicated are checkpointed and rolled back together. Adopting this approach for DSM systems efficiently presents many difficulties [Jana94]. Since processes communicate through read and write accesses to a common address space, rather than through explicit message sends and receives between specific participants, process interactions are less readily identified. The handling of reads and writes that are in progress during checkpoint or recovery is more complicated than the handling of messages in transit due to the coherence protocol. Since the address space and the coherence directory are shared, a naive implementation can propagate checkpoints and recoveries globally leading to a checkpoint or rollback of the entire system. We present an adaptation of coordinated checkpoint and rollback based error recovery for the DSM system [Jana94] which handles these problems efficiently, and prove the correctness of the scheme. Our evaluations indicate that this approach has the potential to reduce checkpointing overhead over previous schemes.

1.3. Organization of this Work

We begin by discussing (in Chapter 2) several issues that are relevant in the implementation of distributed shared memory on a multicomputer. The chapter introduces the problems of consistency, coherency and reliability in the DSM multicomputer, and discusses the performance and storage overhead associated with the implementation in a large scale system. We also briefly describe the salient features of other implementations of distributed shared memory.

Our hierarchical management scheme for DSM is presented in detail in Chapter 3. Chapter 4 describes our implementation of DSM, for execution-driven and trace-driven simulations, in detail. It describes our innovations in the design of the coherence protocol, and the support for DSM in the memory management unit. The results of our performance evalution of the hierarchical scheme using execution-driven and trace-driven simulations are presented and discussed in Chapter 5.

In Chapter 6 we present an adaptation of coordinated checkpoint-rollback based error recovery to a DSM system. We describe the scheme for a DSM system managed at a single level of granularity as well as extensions for a hierarchically managed DSM system. We present results from trace-driven simulations comparing the performance of the coordinated checkpoint-rollback scheme with another scheme proposed earlier.