University of Alberta

Distributed Shared Memory: A Review

by

M. Rasit Eskicioglu and T. Anthony Marsland

Technical Report TR 96–22
September 1996

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

Distributed Shared Memory: A Review*

M. Rasit Eskicioglu and T. Anthony Marsland
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2H1
{rasit,tony}@cs.ualberta.ca

Abstract

In parallel and distributed applications there are two common programming models for interprocess communication: shared memory and message passing. Shared memory has been the standard for tightly-coupled systems (multiprocessors), where the processors have uniform access to a single global memory. Although it is easy to use, memory contention limits the scalability of tightly-coupled systems. On the other hand, message passing has been the major model for loosely-coupled systems (multicomputers), where each processor has a physically separate private memory. Applications that use message passing must move the data back and forth explicitly within programs, making the model burdensome for programmers. In recent years, researchers exploited the shared memory paradigm and studied its applicability to loosely-coupled systems. These efforts resulted in a new abstraction of shared memory on a distributed system that combines the best of the two original models. This concept, commonly known as *Distributed Shared Memory (DSM)*, provides the illusion of a large "shared" memory that extends across machine boundaries. This paper reviews current research in distributed shared memory and related topics.

1 Introduction

Traditionally, shared memory and message passing have been the two programming models for interprocess communication and synchronization in computations performed on a distributed system. Message passing has been the preferred way of handling interprocess communication in loosely-coupled systems, because the computers forming a distributed system do not share physical memory. The message passing model is characterized by data movement among cooperating processes as they communicate and synchronize by *sending* and *receiving* messages. In contrast, tightly-coupled processors primarily use shared memory model since it provides direct support for data sharing.

In recent years, researchers exploited the shared memory paradigm and studied its applicability to loosely-coupled systems. These efforts resulted in the introduction of a new concept that combines the best of the two basic models. This concept, commonly known as *Distributed Shared Memory (DSM)*, refers to the abstraction of memory distributed over several systems, thus

*This work is supported in part by the National Science and Engineering Research Council of Canada grant OPG7902.

providing the illusion of a large "shared" memory. As illustrated in Figure 1, this global memory spans the private memories of the component processors and extends across machine boundaries. DSM allows processes executing on different interconnected processors to share memory by hiding the physical location(s) of data, making the memory *location transparent* to the entire system. An important benefit of this approach is that parallel programs developed for (real) shared memory systems can execute on distributed architectures with no modification.

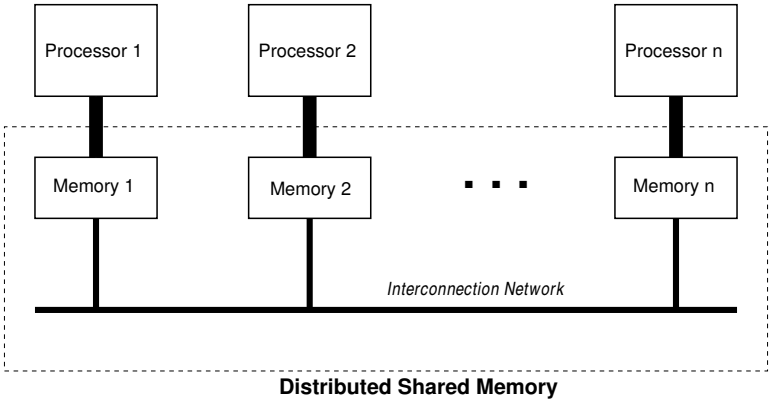

Figure 1: Distributed Shared Memory Abstraction

This paper is motivated by the considerable body of research on DSM in the past decade. Our purpose is to provide an overview of distributed shared memory and to summarize current research in this and related topics. While it is by no means exhaustive, we believe that this review provides a representative picture of the topic covering all important research areas. An earlier work as a comprehensive literature study of DSM is Hellwagner's work [1]. This work aims at identifying basic approaches and specifying desirable architectural support for implementing required abstractions efficiently. In another comparative work, Tam et al. [2] elaborate on the general DSM architecture and compare several systems. Other earlier survey and comparative study on distributed shared memory include [3, 4, 5, 6]. Readers are also referred to the work by Eskicioglu [7] for a more comprehensive list of the literature.

The rest of the paper is organized as follows. Section 2 begins with a brief historical overview and describes early work related to distributed shared memory. We then introduce an informal definition of consistency and briefly discuss common memory consistency models in section 3. Section 4 describes fundamental protocols and algorithms used to provide consistent shared data in a distributed system. Section 5 explores different ways to integrate DSM with the underlying systems. In section 6 and 7, we overview performance studies and other issues surrounding DSM, respectively. Finally, Section 9 elaborates on the current research directions. At the end of each section, we list a representative collection of the literature in each category.

[1] H. Hellwagner. A Survey of Virtually Shared Memory Schemes. Technical Report TUM-19056, Institute for Informatics, Technical University of Munich, Germany, December 1990.

[2] M-C. Tam, J. M. Smith, and D. J. Farber. A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems. *ACM Operating Systems Review*, 24(3), July 1990.

[3] A. Mohindra and U. Ramachandran. A Survey of Distributed Shared Memory in Loosely-coupled Systems. Technical Report GIT-CC-91/01, College of Computing, Georgia Institute of Technology, January 1991.

[4] A. Mohindra and U. Ramachandran. A Comparative Study of Distributed Shared Memory System Design Issues. Technical Report GIT-CC-94/35, College of Computing, Georgia Institute of Technology, August 1994.

[5] S. Raina. Virtual Shared Memory: A Survey of Techniques and Systems. Technical Report CSTR-92-36, Dept. of Computer Science, University of Bristol, 1992.

[6] J. Protic, M. Tomasevic, and V. Milutinovic. A Survey of Distributed Shared Memory Systems. In *Proc. of the 28th Hawaii Int'l Conf. on System Sciences (HICSS-28)*, volume I, pages 74–84, January 1995.

[7] M. R. Eskicioglu. A Comprehensive Bibliography of Distributed Shared Memory. *ACM Operating Systems Review*, 30(1):71–96, January 1996. This bibliography is updated periodically. It is available online at `http://www.cs.ualberta.ca/~rasit/dsmbiblio.html`.

2 Concepts and Origins

Early in 1981, Abramson presented an interesting paper [8] at the Australian Computer Science Conference, considering how to manage large virtual memory from a hardware perspective. At the same conference, another paper by Rosenberg and Keedy [9] addresses the software issues of large virtual memory. Later, a prototype, called MONADS-PC [10], was built based on the concept of a single shared virtual memory. MONADS-PC had a large[1] address space spanning across a network of computers on which all the objects, such as processes, data, and files, are accessed uniformly [11].

An earlier commercial system that employed basic DSM concepts on a network of workstations was the Apollo Domain built on Aegis operating system [13]. The Domain system introduce the *single level store (SLS)* concept, where users share objects in a local area network environment. This approach allows different types of objects such as text files, structured data, and display bitmaps to be transparently mapped into the address spaces of processes. The SLS views the main memory of each node as a cache of objects mapped using a demand paging scheme.

Cheriton [12] proposes somewhat different paradigm, called *problem-oriented shared memory* for building sophisticated distributed applications. This special form of shared memory implements the two fundamental memory operations (fetch and store) in a way that exploits problem-specific semantics. Typically, problem-oriented shared memory provides a particular consistency model and manages this model in behalf of the applications. Cheriton argues that shared memory semantics should be tuned to individual applications as needed and he suggests that studying characteristics of problem semantics would reduce the cost of implementing shared memory. His

[1] The virtual address space of this prototype was 60 bits wide. MONADS-MM, a follow-up prototype, had 128-bit wide address space.

work identifies several generic characteristics, such as relaxed memory schemes allowing temporary "stale data", and describes how they can be exploited to reduce the implementation costs.

[8] D. A. Abramson. Hardware Memory Management of a Large Virtual Memory. In *Proc. of the 4th Australian Computer Science Conf. (ACSC-4)*, pages 1–13, January 1981.

[9] J. Rosenberg and J. L. Keedy. Software Management of a Large Virtual Memory. In *Proc. of the 4th Australian Computer Science Conf. (ACSC-4)*, pages 173–181, January 1981.

[10] J. Rosenberg and D. A. Abramson. MONADS-PC: A Capability Based Workstation to Support Software Engineering. In *Proc. of the 18th Hawaii Int'l Conf. on System Sciences (HICSS-18)*, pages 222–230, January 1985.

[11] D. A. Abramson and J. L. Keedy. Implementing a Large Virtual Memory in a Distributed Computer System. In *Proc. of the 18th Hawaii Int'l Conf. on System Sciences (HICSS-18)*, pages 515–522, January 1985.

[12] D. R. Cheriton. Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems. *ACM Operating Systems Review*, 19(4):26–33, October 1985.

[13] P. J. Leach, P. H. Levine, B. P. Douros, J. Hamilton, D. L. Nelson, and B. L. Stumpf. The Architecture of an Integrated Local Network. *IEEE Journal on Selected Areas in Communications*, SAC-1(5):842–856, November 1983.

In his PhD dissertation [14], Kai Li describes a *software* shared memory abstraction on a loosely-coupled distributed system. He also proposed several algorithms to enforce *strict* memory consistency on this abstraction. IVY [15] is a page-based prototype DSM system implemented on a token-ring network of Apollo workstations. It runs in user mode on top of slightly modified Aegis operating system. We discuss IVY in more detail in Section 5.

[14] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. PhD thesis, Department of Computer Science, Yale University, September 1986.

[15] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proc. of the 1988 Int'l Conf. on Parallel Processing (ICPP'88)*, volume II, pages 94–101, August 1988.

The foundations of distributed shared memory, basically cache coherence and memory management, have been studied since the 1960s. However, detailed research on DSM has been done only during the past decade, and it has since become an active research area resulting in the development of several experimental systems. Many people extended Li's ideas to other such areas as distributed object based systems, language and hardware supported systems. We discuss these research research efforts in Section 5.

3 Consistency Models

Users generally assume when writing programs that "the current value of a variable is determined by the *most recent* write by the program to that variable." This fundamental assumption is also desirable for shared data accessed by programs running on a multiprocessor.

A *consistency model* is used to express the semantics of memory as observed by the programs sharing it. Traditionally, only computer architects designing multiprocessor systems were interested in memory consistency. However, the study of memory consistency became increasingly

popular in recent years and many publications about theoretical aspects appear in the literature. For example, Raynal and Schiper [18] propose a set of formal definitions for several consistency models. Also, Adve and Hill [16] and Sindhu et al. [17] introduce formal specifications of consistency models. The specification of a consistency model provides answers to such questions as: (1) What behavior is expected by the system (i.e., what is the value returned by every read operation performed by a user)? (2) How does the system adhere to the expected consistency of shared data? and (3) What are the constraints imposed on the ordering of shared data accesses performed by two or more processors?

Mosberger [19] classifies the proposed memory consistency models as *uniform* and *hybrid*. Hybrid models employ different ordering constraints depending on the type of memory access, such as ordinary (to shared data) or synchronizing, while the uniform models do not distinguish between the types of memory access. A recent article by Adve and Gharachorloo [20] also gives a detailed overview of various consistency models. The variety of proposals in the literature indicates that there is not a *best* consistency model for distributed systems.

[16] S. V. Adve and M. D. Hill. A Unified Formalization of Four Shared-Memory Models. *IEEE Trans. on Parallel and Distributed Systems*, 4(6):613–624, June 1993.

[17] P. S. Sindhu, J-M. Frailong, and M. Cekleov. Formal Specification of Memory Models. In M. Dubois and S. S. Thakkar, editors, *Scalable Shared Memory Multiprocessors*, pages 25–41. Kluwer Academic Publishers, 1992.

[18] M. Raynal and A. Schiper. A Suite of Formal Definitions for Consistency Criteria in Shared Memories. In *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS'96)*, pages 125–131, September 1996.

[19] D. Mosberger. Memory Consistency Models. *ACM Operating Systems Review*, 27(1):18–26, January 1993. Some correspondence appeared in Volume 27, Number 3 of the same journal. A revised version is available as University of Arizona technical report TR 93/11.

[20] S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. *IEEE Computer*, 29(12):66–76, December 1996.

3.1 Uniform Models

Uniprocessors follow the sequential order specified by the programs (the *program order*) to achieve a simple yet sufficient criteria known as *atomic* (*strict*) *consistency (AC)*. An AC memory guarantees that a read access to a (shared) location always returns the *most recent* value written into that location. However, "*most recent*" is ambiguous in a distributed system, because the results of memory accesses (writes) performed by individual processors may be seen in different order by some other processor(s). For this reason, less strict consistency models are proposed. Among them, the following are the most common:

Sequential Consistency (SC) [21] weakens the AC requirements such that memory access operations may not "take effect" during their execution. The SC model, however, guarantees that "the result of any execution is the same as if the operations of all processors were executed in

some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." Accordingly, a system is *sequentially consistent* if all shared accesses are consistent with *some* global ordering, such that this ordering does not violate individual program orders.

Causal Consistency (CC) [22] is defined as "the agreement of all processors on the order of causally related events (writes)." Basically, Hutto and Ahamad [22] applied Lamport's notion of *potential causality* [23] to distributed shared memory. They argue that events (reads and writes) initiated by individual processors are totally ordered, and that reads of remote writes are related by potential causality. The CC model allows events not causally related (i.e., *concurrent*) to be observed in different orders. Thus, in CC only reads respect the order of causally related writes.

Processor Consistency (PC) [24] states that "the result of any execution is the same as if operations of each individual processor appear [to any other processor] in the sequential order specified by its program." This informal definition is later formalized by Ahamad et al. [25] and eliminated some ambiguities. PC ensures that writes by a given processor are always observed (by the other processors) in the order they are issued. Gharachorloo et al. [26] propose somewhat different consistency model, also called PC. One major difference between the two is that, the latter allows a read of a different location to bypass a write operation in a given program order.

[21] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, C-28(9):690–691, September 1979.

[22] P. W. Hutto and M. Ahamad. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. In *Proc. of the 10th Int'l Conf. on Distributed Computing Systems (ICDCS-10)*, pages 302–311, May 1990.

[23] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[24] J. R. Goodman. Cache Consistency and Sequential Consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.

[25] M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The Power of Processor Consistency (Extended Abstract). In *Proc. of the 5th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'93)*, pages 251–260, June 1993.

[26] K. Gharachorloo, D. E. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. L. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proc. of the 17th Annual Int'l Symp. on Computer Architecture (ISCA'90)*, pages 15–26, May 1990.

3.2 Hybrid Models

Hybrid memory consistency models reduce strictness even further. These models take the advantage of the fact that most parallel and distributed applications enforce higher-level synchronization mechanisms within themselves, thus requiring the enforcement of coherent shared memory only during explicit synchronization operation(s). We summarize the common hybrid models below.

Weak Consistency (WC) [27] separates ordinary shared data access from synchronization data access. The conditions of the WC are:

(1) accesses to synchronization variables are *sequentially consistent*,
(2) accesses to a synchronization variable are only allowed *after* all the previous ordinary shared data accesses are performed, and
(3) accesses to ordinary shared data are only allowed *after* all the previous synchronization accesses are performed.

Basically, a synchronization access is used both to control concurrency between processors and to maintain the integrity of shared data.

Release Consistency (RC) [26] is as an extension of WC, with somewhat relaxed requirements. The conditions for RC are:

(1) accesses to synchronization variables are *processor consistent*,
(2) all previous *acquire* operations must be performed before any access to shared data, and
(3) all previous operations to shared data must be performed before a *release* operation is performed.

Entry Consistency (EC) [28] relates a synchronization variable with each shared datum. In an entry consistent system, processors require consistency of shared data only at the beginning of a critical region.

[27] M. Dubois, C. Scheurich, and F. A. Briggs. Memory Access Buffering in Multiprocessors. In *Proc. of the 13th Annual Int'l Symp. on Computer Architecture (ISCA'86)*, pages 434–442, June 1986.

[28] B. N. Bershad and M. J. Zekauskas. Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors. Technical Report CMU-CS-91-170, School of Computer Science, Carnegie-Mellon University, September 1991.

4 Memory Coherence Protocols and Algorithms

DSM provides a global view of all memories which should be maintained consistently according to the memory model used. Coherence protocols, similar to cache coherence protocols found in multiprocessors, are used to implement the required consistency semantics. The protocol is usually straightforward if there is no replication among the shared data. In this case, the coherence can easily be achieved by serializing the accesses to the data through the underlying network on each processor. However, this method severely reduces the major advantages, namely scalability and parallelism, of DSM. Thus, data replication is often required. Unfortunately, data replication complicates the coherence protocols, since the protocols should also deal with multiple copies of the shared data.

There are two common protocols to handle replication: *write-invalidate* and *write-update*.

- The **Write-invalidate protocol** broadcasts an invalidation request when a replica is modified by a processor. Hence, it allows multiple read-only copies and one write-only copy to exist. However, before a write operation is actually performed on the write-only copy, all the other

(read-only) copies must be invalidated. This protocol is also known as the *multiple-readers-single-writer (MRSW)* protocol.

- The **Write-update protocol** broadcasts the new value of the data when a replica is modified by a processor. Since the write operations are immediate, this protocol allows multiple write-only copies of shared data, as well as multiple read-only copies. Because of this characteristics, it is also known as the *multiple-reader-multiple-writer (MRMW)* protocol.

Synchronized access to shared data is achieved by low level machine instructions such as *Test-and-Set* in shared memory multiprocessors, In DSM systems, however, the use of such instructions on arbitrary memory accesses is meaningless. A general solution is to provide the users with high level synchronization primitives, such as locks and barriers, implemented using message passing. Also, applications may synchronize only when necessary (for example, to indicate the completion of a computation.)

In addition to keeping the shared memory consistent, a DSM system should also provide algorithms to locate and access shared data. Stumm and Zhou [29] categorize such algorithms based on whether the data are migratory, replicated, or both as follows:

- **Central-server algorithm**. Shared data resides in a fixed and known location and is maintained by a server. The users (clients) of the shared data send requests to the server and the server responds to those requests. Although this algorithm is quite simple, it has a potential bottleneck where the server node may become overloaded by frequent requests.
- **Migration algorithm:** Shared data relocate to the requesting nodes as they are accessed. The shared data can also be grouped into larger units to reduce communication costs, allowing neighboring data to be accessed locally. If the data block is not local, a client broadcasts a "location request" message. Once it is located, the client sends a "migrate request" message to the current holder to receive the data block. However, the communication costs can still be high if the application exhibits poor locality of reference. Also, this two-phase algorithm causes unnecessary traffic on the network.
- **Read-replication algorithm**. Shared data is replicated with read operations. This usually reduces the communication overhead, since multiple read operations can be performed simultaneously. However, to maintain consistency for a write operation, the requester must first multicast an invalidate message to the holders of the replicas. This algorithm basically follows the write-invalidate protocol.
- **Full-replication algorithm**. This goes one step further and allows multiple writable copies of the data blocks, but complicates the consistency maintenance of shared data. A global "sequencer" controls accesses to the shared data to ensure consistency.

[29] M. Stumm and S. Zhou. Algorithms Implementing Distributed Shared Memory. *IEEE Computer*, 23(5):54–64, May 1990.

5 System Models and Architectures

DSM implementations are integrated in parallel and distributed systems at different levels, presenting the users with an abstraction of global shared memory (Figure 2). Researchers have proposed three ways to provide DSM: hardware enhancements, operating system primitives and system libraries, and language and application level mechanisms. These implementations, however, are not mutually exclusive. Wilson et al. [30] argue that a hybrid approach provides a combination of the advantages of the individual abstractions.

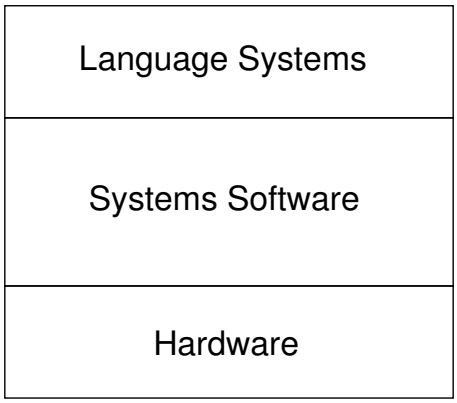

Figure 2: Levels of Integration

[30] A. W. Wilson Jr., R. P. LaRowe Jr., and M. J. Teller. Hardware Assist for Distributed Shared Memory. In *Proc. of the 13th Int'l Conf. on Distributed Computing Systems (ICDCS-13)*, pages 246–255, May 1993.

[31] D. Chaiken and A. Agarwal. Software-Extended Coherent Shared Memory: Performance and Cost. In *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA'94)*, pages 314–324, April 1994.

The following sections introduce different levels of integration and give some examples of such implementations.

5.1 Hardware Implementations

Providing hardware support to DSM has been exploited in several ways. Some systems explore the idea of enhancing the networking capabilities of the loosely-coupled systems. For example, the development of MemNet [32] is based on the observation that the network is always treated as an I/O device by the communication protocols. MemNet is a shared memory local area network, based on a high-speed token ring, where the local network appears as memory in the physical address space of each processor. Capnet [33] extends this idea to a wider domain, namely wide area networks, whereas the DASH system [34] aims at building a scalable high performance architecture with single address space and coherent caches. Accordingly, the design of DASH

combines the programmability of shared memory machines and the scalability of message passing machines.

The Data Diffusion Machine (DDM) [35] is a new architecture that supports a cache-only memory model. The DDM views the entire memory of the system as a large cache of its virtual address space. PLUS [36] is a multiprocessor architecture that uses distributed memories with hardware supported memory coherence and synchronization mechanisms. The PLUS architecture is tailored to the fast and efficient execution of a single multi-threaded process on each processor. Galactica Net [37] is a hardware assisted DSM architecture. The coherency of the shared memory is achieved through the use of a special interface module, called *Galactica Net Interface Module (GIM)*.

Other hardware assisted DSM architectures include research machines such as MIT's Alewife [38], SUNY–Stony Brook's Sesame [39], Rice University's Willow [40], the Wisconsin Multicube [41], and commercial systems such as BBN Butterfly and Kendall Square Research's KSR1. These architectures generally employ hierarchical designs. With the exception of Galactica Net, Sesame and Alewife, all the hardware solutions rely mainly on complex hardware strategies to reduce shared memory access times. Yet, most of the hardware implementations are supported to some extend by high-level software techniques to reduce the amount of communication overhead among the involved processors.

The following are some currently active projects related to hardware implementations of DSM: FLASH is a scalable multiprocessor capable of supporting a variety of communication models [42]; SHRIMP is a parallel multicomputer built from off-the-shelf Pentium PCs interconnected to a routing network with an in-house network interface [43]; DICE is a modular and scalable platform architecture based on a cache-only distributed memory scheme [44]; IA-COMA is a scalable shared memory multiprocessor organized as a flat cache-only memory architecture [45]; S3.mp is a commercial scalable multiprocessor prototype based on cache-coherent non-uniform memory access architecture [46]; and Avalanche is a project aiming at building a scalable parallel computing platform from inexpensive components [47].

[32] G. S. Delp, A. S. Sethi, and D. J. Farber. An Analysis of MemNet: An Experiment in High-Speed Shared-Memory Local Networking. In *Proc. of the ACM Symp. on Communications Architectures, Protocols and Applications (SIGCOMM'88)*, pages 165–174, August 1988.

[33] M-C. Tam and D. J. Farber. CapNet–An Approach to Ultra High Speed Network. In *Proc. of the IEEE Int'l Conf. on Communications (ICC'90)*, pages 955–961, April 1990.

[34] D. E. Lenoski, J. Ludon, K. Gharachorloo W-D. Weber, A. Gupta, J. L. Hennessy, M. Horowitz, and M. S. Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[35] D. H. D. Warren and S. Haridi. Data Diffusion Machine—A Scalable Share Virtual Memory Multiprocessor. In *Proc. of the Int'l Conf. on Fifth Generation Computer Systems (ICOT'88)*, pages 943–952, 1988.

[36] R. Bisiani and M. Ravishankar. PLUS: A Distributed Shared-Memory System. In *Proc. of the 17th Annual Int'l Symp. on Computer Architecture (ISCA'90)*, pages 115–124, May 1990.

[37] A. W. Wilson Jr., T. H. Probert, T. Lane, and B. Fleischer. Galactica Net: An Architecture for Distributed Shared Memory. In *Proc. of the 1991 GOMAC*, pages 513–516, November 1991.

[38] A. Agarwal, B-H. Lim, D. Kranz, and J. Kubiatowicz. APRIL: A Processor Architecture for Multiprocessing. In *Proc. of the 17th Annual Int'l Symp. on Computer Architecture (ISCA'90)*, pages 14–110, May 1990.

[39] L. D. Wittie, G. Hermannsson, and A. Li. Eager Sharing for Efficient Massive Parallelism. In *Proc. of the 1992 Int'l Conf. on Parallel Processing (ICPP'92)*, pages 251–255, August 1992.

[40] J. K. Bennett, S. Dwarkadas, J. A. Greenwood, and E. Speight. Willow: A Scalable Shared Memory Multiprocessor. In *Proc. of Supercomputing'92*, pages 336–345, November 1992.

[41] J. R. Goodman and P. J. Woest. The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor. In *Proc. of the 15th Annual Int'l Symp. on Computer Architecture (ISCA'88)*, pages 422–431, May 1988.

[42] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. L. Hennessy. The Stanford FLASH Multiprocessor. In *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA'94)*, pages 302–313, April 1994.

[43] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA'94)*, pages 142–153, April 1994.

[44] G. Lee. An Assessment of COMA Multiprocessors. In *Proc. of the 9th Int'l Parallel Processing Symp. (IPPS'95)*, pages 388–392, April 1995.

[45] J. Torrellas and D. Padua. The Illinois Aggressive Coma Multiprocessor Project (I-ACOMA). In *Proc. of the 6th Symp. on the Frontiers of Massively Parallel Computing (Frontiers'96)*, October 1996.

[46] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, D. Lee, and M. Parkin. The S3.mp Scalable Shared Memory Multiprocessor. In *Proc. of the 27th Hawaii Int'l Conf. on System Sciences (HICSS-27)*, volume I, pages 144–153, January 1994.

[47] J. B. Carter, A. Davis, R. Kuramkote, C-C. Kuo, L. B. Stoller, and M. Swanson. Avalanche: A Communication and Memory Architecture for Scalable Parallel Computing. In *Proc. of the Fifth Workshop on Scalable Shared Memory Multiprocessors*, June 1995.

5.2 Software Implementations

During the past decade, several prototypes have been built that provide a DSM abstraction at the system level. System level implementations usually integrate the DSM as a region of virtual address space in the participating programs using the virtual memory management system of the underlying operating system.

Li's shared virtual memory [14] provides users with an interface similar to the memory address space on a multiprocessor architecture. Later, he expanded this idea to other architectures and developed a prototype called Shiva on a hypercube [48].

Munin [49] is a runtime system and a server mechanism to allow programs written for shared memory multiprocessors to be executed efficiently in a distributed memory environment. The runtime system handles faults, threads, and synchronization mechanisms and provides support for multiple consistency protocols [50], while the server mechanism handles the correct mapping of shared segments into local memories. DSM is implemented as a two-level mechanism in Clouds [51]: the *Distributed Shared Memory Controller (DSMC)* [52] provides data transfer and synchronization primitives for supporting the abstraction of a global distributed shared memory, while the *DSM partition* provides the kernel with the ability to create, destroy, activate, deactivate segments, page-in, page-out portions of segments, and the semaphore operations. The DSMC is also proposed as a hardware component [53].

METHER [54] is a DSM system that runs on a network of Sun workstations integrated with the SunOS operating system. It defines a special address space, called *Mether Address Space*, which is a collection of fixed locations on the virtual address space of each workstation. The network file system (NFS) of SunOS operating system was later modified to support virtual shared memory [55]. MIRAGE [56] extends SYSTEM V shared memory semantics to a network of computers. Users organize and access their shared data through segments which are used to store data. The original prototype of MIRAGE was developed on a network of three VAX 11/750 minicomputers. Later, this prototype was ported to a network of IBM PS/2 personal computers, and is now called MIRAGE+ [57]. In the CHOICES project [58], the distributed shared memory is implemented using object-oriented techniques [59]. The CHOICES DSM system is composed of two parts: an extension of CHOICES virtual memory class hierarchy that provides page fault handling and maintains memory coherence, and a network protocol that specifies the communications protocol among nodes that shares memory to maintain memory coherence.

Newer research on distributed shared memory also extends to advanced software projects. The Coherent Virtual Machine (CVM) provides multiple protocol support, extensibility, and fault tolerance to the applications [60]. DiSOM is built as a software layer on the operating systems of heterogeneous multicomputers [61]. KOAN is embedded in the operating system of a iPSC/2 hypercube multicomputer [62]. The MILLIPEDE project aims at developing a distributed shared memory environment for parallel programming [63]. QUARKS is a portable system supporting multiple consistency protocols and multi-threading [64]. Phosphorus [65] and Adsmith [66] both sit on top of the popular message passing kernel, PVM [67]. PAMS [68] and TREADMARKS [69] are the only two commercial DSM systems available today.

[48] K. Li and R. Schaefer. Shiva: An Operating System Transforming A Hypercube into a Shared-Memory Machine. Technical Report CS-TR-217-89, Dept. of Computer Science, Princeton University, April 1989.

[49] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Shared Memory for Distributed Memory Multiprocessors. Technical Report COMP TR89-91, Dept. of Computer Science, Rice University, April 1989.

[50] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence. In *Proc. of the Second ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'90)*, pages 168–177, March 1990.

[51] U. Ramachandran, M. Ahamad, and M. Y. A. Khalidi. Coherence of Distributed Shared Memory: Unifying Synchronization and Data Transfer. In *Proc. of the 1989 Int'l Conf. on Parallel Processing (ICPP'89)*, volume II, pages 160–169, August 1989.

[52] U. Ramachandran and M. Y. A. Khalidi. Programming with Distributed Shared Memory. In *Proc. of the 13th Annual Int'l Computer Software and Applications Conf. (COMPSAC'89)*, pages 176–183, September 1989.

[53] A. Mohindra. *Issues in the Design of Distributed Shared Memory Systems*. PhD thesis, College of Computing, Georgia Institute of Technology, May 1993.

[54] R. G. Minnich and D. J. Farber. The Mether System: Distributed Shared Memory for SunOS 4.0. In *Proc. of the 1989 Summer USENIX Conference*, pages 51–60, June 1989.

[55] R. G. Minnich and D. V. Pryor. Mether: Supporting the Shared Memory Model on Computing Clusters. In *Proc. of the 38th IEEE Int'l Computer Conf. (COMPCON Spring'93)*, pages 558–567, February 1993.

[56] B. D. Fleisch. Distributed Shared Memory in a Loosely Coupled Distributed System. In *Proc. of the ACM SIGCOMM'87 Workshop on Frontiers in Computer Communications Technology*, pages 317–327, August 1987.

[57] B. D. Fleisch, R. L. Hyde, and N. C. Juul. Moving Distributed Shared Memory to the Personal Computer: The MIRAGE+ Experience. Technical Report UCR-CS-93-6, Dept. of Computer Science, University of California at Riverside, June 1993.

[58] Roy Campbell, Gary Johnston, and Vincent Russo. Choices—Class Hierarchical Open Interface for Custom Embedded Systems. *ACM Operating Systems Review*, 21(3), July 1987.

[59] G. M. Johnston and R. H. Campbell. An Object-Oriented Implementation of Distributed Virtual Memory. In *Proc. of the Symp. on Experiences with Distributed and Multiprocessor Systems*, pages 39–57, October 1989.

[60] P. Keleher. The Relative Importance of Concurrent Writers and Weak Consistency Models. In *Proc. of the 16th Int'l Conf. on Distributed Computing Systems (ICDCS-16)*, pages 91–98, May 1996.

[61] P. Guedes and M. Castro. Distributed Shared Object Memory. In *Proc. of the 4th Workshop on Workstation Operating Systems (WWOS-IV)*, pages 142–149, October 1993.

[62] Z. Lahjomri and T. Priol. KOAN: A Shared Virtual Memory for iPSC/2 Hypercube. In *Proc. of the 2nd Joint Int'l Conf. on Vector and Parallel Processing (CONPAR'92)*, pages 441–452, September 1992.

[63] R. Friedman, G. Maxim, A. Itzkovitz, and A. Schuster. MILLIPEDE: Easy Parallel Programming in Available Distributed Environments. In *Proc. of the Second Int'l Euro-Par Conf.*, pages 84–87, August 1996.

[64] J. B. Carter, D. Khandekar, and L. Kamb. Distributed Shared Memory: Where We Are and Where We Should Be Headed? In *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 119–122, May 1995.

[65] I. Demeure, R. Cabrera-Dantart, and P. Meunier. Phosphorus: A Distributed Shared Memory System on Top of PVM. In *Proc. of EUROMICRO'95*, pages 269–273, September 1995.

[66] W-Y. Liang, C-T. King, and F. Lai. Adsmith: An Efficient Object-Based Distributed Shared Memory on PVM. In *Proc. of the 2nd Int'l Symp. on Parallel Architectures, Algorithms, and Networks (I-SPAN'96)*, pages 173–179, June 1996.

[67] V. S. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

[68] Myrias Computer Technologies, Inc. *Parallel Application Management System (PAMS) V2*, 1995.

[69] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, January 1994.

5.3 Language Support

The highest level of integration provides distributed shared memory abstraction either by extending languages through run time libraries or by providing new language constructs. *Tuple Space* is a novel synchronization mechanism developed for Linda [70], a parallel language based on the notion of distributed shared memory. Similar to a record construct in Pascal, Tuple Space is a global memory containing *tuples*. Unlike the other conceptual shared memory systems, however, it is addressed associatively (by contents). Amber [71] is a programming system that permits application programs to view a homogeneous network of computers as an integrated multiprocessor. Orca [72], however, views DSM as a collection of shared objects of variable size and structures.

Current DSM projects that extend languages (mainly the C language) include *Cilk* [73], *Filaments* [74], locust [75], and SAM [76]. *Cilk* is a multi-threaded runtime system that provides a relaxed consistency, called *DAG-consistency* [73]. *Filaments* is a software package that supports fine-grain parallelism efficiently on shared- and distributed-memory machines [74]. locust is an object-based DSM that uses compile-time data dependency information to improve application

performance by addressing issues such as false sharing, coherence maintenance, data prefetching, and affinity scheduling [75]. Finally, SAM, which also uses PVM as the underlying message passing system, is a portable runtime system that provides a global name space and automatic caching of shared data [76].

[70] S. Ahuja, N. Carriero, and D. Gelernter. Linda and Friends. *IEEE Computer*, 19(8):26–34, August 1986.

[71] J. S. Chase, F. G. Amador, E. D. Lazowska, H. M. Levy, and R. J. Littlefield. The Amber System: Parallel Programming on a Network of Multiprocessors. In *Proc. of the 12th ACM Symp. on Operating Systems Principles (SOSP-12)*, pages 147–158, December 1989.

[72] H. E. Bal. *The Shared Data-Object Model as a Paradigm for Programming Distributed Systems*. PhD thesis, Free University, The Netherlands, 1989.

[73] R. D. Blumofe, M. Frigo, C. F. Joerg, C. E. Leiserson, and K. H. Randall. Dag-Consistent Distributed Shared Memory. In *Proc. of the 10th Int'l Parallel Processing Symp. (IPPS'96)*, pages 132–141, April 1996.

[74] V. W. Freeh, D. K. Lowenthal, and G. R. Andrews. Distributed Filaments: Efficient Fine-Grain Parallelism on a Cluster of Workstations. In *Proc. of the 1st Symp. on Operating Systems Design and Implementation (OSDI'94)*, pages 201–213, November 1994.

[75] T-C. Chiueh and M. Verma. A Compiler-Directed Distributed Shared Memory System. In *Proc. of the 9th ACM-SIGARCH Int'l Conf. on Supercomputing*, pages 77–86, July 1995.

[76] D. J. Scales and M. S. Lam. The Design and Evaluation of a Shared Object System for Distributed Memory Machines. In *Proc. of the 1st Symp. on Operating Systems Design and Implementation (OSDI'94)*, pages 101–114, November 1994.

6 Performance Evaluation and Analysis

Performance evaluation of distributed shared memory systems (both hardware and software approaches) is not an easy task. Performance aspects of hardware DSM architectures are evaluated on simulators of the hardware [77]. Software memory consistency models are evaluated either by simulation [78] or by other tools, such as petri nets [79, 80].

Keleher et al. [81] evaluate three implementations of software-based release consistent protocols. Bodin et al. report the performance of a test suite on KOAN SVM [82]. Levelt et al. [83] compare a language based DSM with a IVY–like system level implementation. Sun and Zhu [84] propose "generalized speedup" as a new performance metric. Also, some work was done measuring the performance of parallel applications that run on distributed shared memory systems [85, 86, 87].

[77] R. Chandra, K. Gharachorloo, V. Soundararajan, and A. Gupta. Performance Evaluation of Hybrid Hardware and Software Distributed Shared Memory Protocols. In *Proc. of the 8th ACM-SIGARCH Int'l Conf. on Supercomputing*, pages 274–288, July 1994.

[78] K. Gharachorloo, A. Gupta, and J. L. Hennessy. Performance Evaluation of Memory Consistency Models for Shared Memory Multiprocessors. In *Proc. of the 4th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 245–257, April 1991.

[79] C. Lindemann and F. Schoen. Evaluating Sequential Consistency in a Virtually Shared Memory System by Deterministic and Stochastic Petri Nets. In *Proc. of the 1st Int'l Workshop on Modeling, Analysis, and Simulation of Computers and Telecommunication Systems (MASCOTS'93)*, pages 63–68, January 1993.

[80] Y-K. Chong and K. Hwang. Performance Analysis of Four Memory Consistency Models for Multithreaded Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 6(10):1085–1099, October 1995.

[81] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. An Evaluation of Software-Based Release Consistent Protocols. *Journal of Parallel and Distributed Computing*, 29(2):126–141, September 1995.

[82] F. Bodin and T. Priol. Overview of the KOAN Programming Environment for the iPSC/2 and Performance Evaluation of the BECAUSE Test Program 2.51. *Future Generation Computer Systems*, 10(4):391–401, November 1994.

[83] W. G. Levelt, M. F. Kaashoek, H. E. Bal, and A. S. Tanenbaum. A Comparison of Two Paradigms for Distributed Shared Memory. *Software—Practice and Experience*, 22(11):985–1010, November 1992. Also available as Free University of the Netherlands, Computer Science Department technical report IR-221.

[84] X-H. Sun and J. Zhu. Performance Considerations of Shared Virtual Memory Machines. *IEEE Trans. on Parallel and Distributed Systems*, 6(11):1185–1194, November 1995.

[85] S. Dwarkadas, A. Schaffer, R. W. Cottingham, A. L. Cox, P. Keleher, and W. Zwaenepoel. Parallelization of General Linkage Analysis Problems. *Human Heredity*, 44:127–141, July 1994.

[86] R. G. Minnich and D. V. Pryor. A Radiative Heat Transfer Simulation on a SPARCStation Farm. In *Proc. of the First IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-1)*, pages 124–132, September 1992.

[87] P. Wang. Solving A Highly Irregular Problem on Distributed Shared Memory Systems. In *Proc. of the 1995 ICPP Workshop on Challenges for Parallel Processing*, August 1995.

7 Related Issues

Over the past few years, the research on DSM has been expanded to include some other related issues, such as heterogeneity, recoverability, and fault-tolerance.

The AGORA [88] and MERMAID [89] projects aim at extending DSM to heterogeneous system environments. As a language level DSM, AGORA supports multilanguage modules running on heterogeneous machines, by providing a set of access functions to create and manipulate shared data structures. These functions can be accessed by different languages such as C and CommonLisp. MERMAID, however, is implemented as a user-level DSM with minor operating system kernel modifications. Although implemented in different languages on different machines, MERMAID only supports the C language. Both research efforts conclude that the major problem is data conversion and that for values at limits of range, different representations of floating point numbers make the conversion impossible. Zhou et al. [90] argue that the number of different machines accommodating heterogeneous DSM can be extended arbitrarily at the cost of providing separate conversion routines for basic data types on each pair of machines. The development and coding of these conversion routines is a one-time only effort and is transparent to the users. On the other hand, inter-processor (network) data formats can be defined to eliminate this effort. However, this would increase runtime conversion overhead, as the data would have to be converted twice (first from sender's format into the standard format and then from the standard format into the receiver's format) each time they are transported.

[88] R. Bisiani and A. Forin. Multilanguage Parallel Programming of Heterogeneous Machines. *IEEE Transactions on Computers*, 37(8):930–945, August 1988.

[89] S. Zhou, M. Stumm, and T. McInerney. Extending Distributed Shared Memory to Heterogeneous Environments. In *Proc. of the 10th Int'l Conf. on Distributed Computing Systems (ICDCS-10)*, May 1990.

[90] S. Zhou, M. Stumm, K. Li, and D. Wortman. Heterogeneous Distributed Shared Memory. *IEEE Trans. on Parallel and Distributed Systems*, 3(5):540–554, September 1992. Also available as University of Toronto, Computer Systems Research Institute technical report TR-CSRI-244.

Another problem that has recently drawn research interest is the recoverability of shared data after processor failures. Wu and Fuchs [91] examine the problem of rollback recovery in distributed shared memory environments using checkpointing and a twin-page disk storage technique. Their checkpointing scheme is transparent to users and is integrated into the coherence protocol. Janssens and Fuchs [92] introduce new algorithms for checkpointing that take advantage of relaxed consistency memory models. Two other works take different approaches to address the recoverability of distributed shared memory: Kermarrec et al. [93] propose a recovery technique that extends Li's static distributed coherency management mechanism, Kim and Vaidya [94], in contrast, propose a competitive update protocol.

[91] K-L. Wu and W. K. Fuchs. Recoverable Distributed Shared Memory. *IEEE Transactions on Computers*, 39(4):460–469, April 1990.

[92] B. Janssens and W. K. Fuchs. Relaxing Consistency in Recoverable Distributed Shared Memory. In *Proc. of the 23rd Annual Int'l Symp. on Fault-Tolerant Computing (FTCS-23)*, pages 155–163, 1993.

[93] A-M. Kermarrec, G. Cabillic, A. Gefflaut, C. Morin, and I. Puaut. A Recoverable Distributed Shared Memory Integrating Coherence and Recoverability. In *Proc. of the 25th Annual Int'l Symp. on Fault-Tolerant Computing (FTCS-25)*, pages 289–298, June 1995.

[94] J-H. Kim and N. H. Vaidya. Recoverable Distributed Shared Memory Using the Competitive Update Protocol. In *Proc. of the 1995 IEEE Pacific Rim Conf. on Fault Tolerant Systems (PRFTS'95)*, December 1995.

Stumm and Zhou [95] extend four basic DSM algorithms making them resilient to system faults. They argue that host failures are not frequent and the failures in most cases are independent of each other (except for power failure which might affect many hosts), thus tolerating a single host failure is sufficient for most applications. The study shows that the extended versions of the central-server and the full-replication algorithms do not introduce significant additional overhead. However, the overhead introduced by the migration and the read-replication algorithms may be substantial and reduce the performance of these algorithms dramatically, depending on the access patterns of the applications. Brown and Wu [96] present a new set of fault-tolerant algorithms. Banatre et al. [97] describe a reliable DSM built on top of the Mach kernel.

[95] M. Stumm and S. Zhou. Fault Tolerant Distributed Shared Memory. In *Proc. of the Second IEEE Symp. on Parallel and Distributed Processing*, pages 719–724, December 1990.

[96] L. Brown and J. Wu. Snooping Fault-Tolerant Distributed Shared Memories. *The Journal of Systems and Software*, 29(2):149–165, May 1995.

[97] M. Banatre, G. Muller, B. Rochat, and P. Sanchez. A Reliable Distributed Virtual Memory on Top of the MACH Kernel. In *OSF Micro Kernel Applications Workshop*, November 1990.

Some researchers have investigated the garbage collection issue in distributed shared memory

[98, 99, 100]. Other issues, though not extensively studied, include scalability [101] and use of transactions [102] in DSM.

[98] R. Kordale, M. Ahamad, and J. Shilling. Distributed/Concurrent Garbage Collection in Distributed Shared Memory Systems. In *Proc. of the Third Int'l Workshop on Object Orientation in Operating Systems (IWOOOS'93)*, pages 51–60, December 1993.

[99] M. Shapiro and P. Ferriera. Larchant-RDOSS: A Distributed Shared Persistent Memory and Its Garbage Collector. In *Proc. of the 9th Int'l Workshop on Distributed Algorithms (WDAG'95)*, pages 198–214, September 1995.

[100] W. M. Yu and A. L. Cox. Conservative Garbage Collection on Distributed Shared Memory Systems. In *Proc. of the 16th Int'l Conf. on Distributed Computing Systems (ICDCS-16)*, pages 402–410, May 1996.

[101] S. Murer and P. Farber. A Scalable Distributed Shared Memory. In *Proc. of the 2nd Joint Int'l Conf. on Vector and Parallel Processing (CONPAR'92)*, pages 453–466, September 1992.

[102] P. Bodorik, F. I. Smith, and D. J. Lewis. Transactions in Distributed Shared Memory Systems. In *Proc. of the 8th Int'l Conf. on Data Engineering*, pages 480–487, February 1992.

8 Conclusion

We have reviewed the current research in distributed shared memory. Admittedly, substantial research progress has been achieved over the past decade and more work is underway. The wide variety of research in the field suggests that no one technique developed so far is alone sufficient to provide an efficient implementation. Techniques solely based on hardware suffer from the lack of proper technology, which can be partially eliminated by the use of sophisticated software methods. On the other hand, software approaches alone usually have communication latencies regardless of the techniques used. This overhead can usually be reduced by language-level primitives. Language extensions then increases the complexity of programming.

We hope that this review will provide a solid starting point and a reference for interested researchers. More research on DSM will be following. For example, expansion of the scope of DSM, which is currently almost exclusively directed at large scientific applications, provision of debugging and performance tuning tools for DSM, and better integration of the DSM systems with the rest of the programming environments, are much needed [64].