International Journal of Advanced Research in Engineering and Applied Sciences
ISSN: 2278-6252

A REVIEW OF DISTRIBUTED SHARED MEMORY

Pankaj Sharma*

Naveen Malik*

Naeem Akhtar*

Rahul*

Hardeep Rohilla*

Abstract: *Distributed shared memory (DSM) systems have attracted considerable research efforts recently, since they combine the advantages of two different computer classes: shared memory multiprocessors and distributed systems. The most important one is the use of shared memory programming paradigm on physically distributed memories. In the first part of this paper, one possible classification taxonomy, which includes two basic criteria and a number of related characteristic, is proposed and described. According to the basic classification criteria-implementation level of DSM mechanism--systems are organized into three groups: hardware, software, and hybrid DSM implementations. The second part of the paper represents an almost exhaustive survey of the existing solutions in an uniform manner, presenting their DSM mechanisms and issues of importance for various DSM systems and approaches.*

Software DSM systems can be implemented in an operating system (OS), or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the OS, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users.

*Student, CSE, Dronachraya College of Engineering, Gurgaon

HISTORY:-

Memory mapped files started in the MULTICS operating system in the 1960s. One of the first DSM implementations was Apollo. One of the first system to use Apollo was Integrated shared Virtual memory at Yale (IVY). DSM developed in parallel with shared-memory multiprocessors.

INTRODUCTION:-

Distributed Shared Memory (DSM), in Computer Architecture is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term **shared** does not mean that there is a single centralized memory but **shared** essentially means that the address space is shared (same physical address on two processors refers to the same location in memory). **Distributed Global Address Space** (**DGAS**), is a similar term for a wide class of software and hardware implementations, in which each node of a cluster has access to shared memory in addition to each node's non-shared private memory.

Software DSM systems can be implemented in an operating system (OS), or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the OS, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, Software DSM systems implemented at the library or language level are not transparent and developers usually have to program differently. However, these systems offer a more portable approach to DSM system implementation.

Software DSM systems also have the flexibility to organize the shared memory region in different ways. The page based approach organizes shared memory into pages of fixed size. In contrast, the object based approach organizes the shared memory region as an abstract space for storing shareable objects of variable sizes. Another commonly seen implementation uses a tuple space, in which the unit of sharing is a tuple.

Shared memory architecture may involve separating memory into shared parts distributed amongst nodes and main memory; or distributing all memory between nodes. A coherence protocol, chosen in accordance with a consistency model, maintains memory coherence

Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates

to what appears to be ordinary memory within their address space. However, an underlying runtime system ensures transparently that processes executing at different computers observe the updates made by one another. It is as though the processes access a single shared memory, but in fact the physical memory is distributed. The main point of DSM is that it spares the programmer the concerns of message passing when writing applications that might otherwise have to use it. DSM is primarily a tool for parallel applications or for any distributed application or group of applications in which individual shared data items can be accessed directly. DSM is in general less appropriate in client-server systems, where clients normally view server-held resources as abstract data and access them by request (for reasons of modularity and protection). However, servers can provide DSM that is shared between clients. For example, memory-mapped files that are shared and for which some degree of consistency is maintained are a form of DSM. (Mapped files were introduced with the MULTICS operating system [Organick 1972].) Message passing cannot be avoided altogether in a distributed system: in the absence of physically shared memory, the DSM runtime support has to send updates in messages between computers. DSM systems manage replicated data: each computer has a local copy of recently accessed data items stored in DSM, for speed of access. The problems of implementing DSM are related to those discussed in Chapter 15, as well as those of caching shared files discussed in Chapter 8.

One of the first notable examples of a DSM implementation was the Apollo Domain file system. 1983], in which processes hosted by different workstations share files by mapping them simultaneously into their address spaces. This example shows that distributed shared memory can be persistent. That is, it may outlast the execution of any process or group of processes that accesses it and be shared by different groups of processes over time.

Examples of such systems include:

- Kerrighed
- OpenSSI
- MOSIX
- TreadMarks

1. DSM CLASSIFICATION:-

In order to provide a wide and extensive overview in the field of DSM, possible platforms for classification and a set of relevant parameters that must be considered in DSM design are

proposed. The selection of classification criteria can be taken conditionally, since some of the parameters could also be adopted as the platform for classification. Our choice of classification criteria relies on the possibility to classify all existing systems into the appropriate non-overlapping subsets of systems with common general advantages and drawbacks.

The criterion: DSM implementation level

Types:

1. Hardware

2. Software

2.1. Operating system

2.1.1. Inside the kernel

2.1.2. Outside the kernel

2.2. Runtirne library routines

2.3. Compiler-inserted primitives

3. Hardware/software combination

The level of DSM implementation affects both the programming model and the overall system performance. While the hardware solutions bring the total transparency to the programmer, and achieve very low access latencies, software solutions can better exploit the application behavior and represent the ideal polygon to experiment with new concepts and algorithms. As the consequence, the number of software DSM systems presented in the open literature is considerably higher, but the systems intending to become commercial products and standards are mostly hardware-oriented.

Parameters closely related to the DSM implementation level

Some important characteristics of the system are often (but not necessarily) closely related, or even determined by this criterion.

Architectural configuration of the system affects the system performance, since it can offer or restrict a good potential for parallel processing of requests related to the DSM management. It also strongly affects the scalability. Since a system applying a DSM mechanism is usually organized as a set of clusters connected by an interconnection network, architectural parameters include:

a) Cluster configuration (single/multiple processors, with/without, shared&ivate, single/multiple level caches, etc.)

b) Interconnection network (bus hierarchy, ring, mesh, hypercube, specific LAN, etc.)

Cluster configuration is usually very important for the hardware-oriented proposals that integrate the mechanisms of cache coherence on the lower level with the DSM mechanisms on the higher level of the system organization, or even store all shared data in large caches. Cluster configuration is mostly transparent for software solutions, It includes the memory organization and the placement of directory, as well. Almost all types of interconnection networks found in multiprocessors and distributed systems have also been used in DSM systems, The majority of software-oriented DSM systems were actually build on the top of Ethernet, although some of the solutions tend to be architecture independent and portable to various platforms. On the other hand, topologies such as bus hierarchy or mesh are typical for hardware solutions. The choice of topology can be also very important for the implementation of DSM algorithm, since it affects the possibility and cost of broadcast and multicast transactions. Shared data organization represents the global layout of shared address space, as well as the size and organization of data

items in it, and can be distinguished as:

a) Structure of shared data (non structured or structured into objects, language types. etc.)

b) Granularity of coherence unit (word, cache block, page, complex data structure, etc.)

The impact of this organization to the overall system performance is closely related to the locality of data access.

Hardware solutions always &al with non-structured data objects (typically cache blocks), while many software implementations tend to use data items that represent logical entities, in order to take advantage of the locality naturally expressed by the application. On the other hand, some software solutions, based on virtual memory mechanisms, organize data in larger physical blocks (pages), counting on the coarse-grain sharing.

The second criterion: DSM algorithm

Types:

1. SRSW (Single Reader/Single Writer)

1.1. Without migration

1.2. With migration

2. MRSW (Multiple Reader/Single Writer)

3. MRAM' (Multiple Reader Multiple Writer)

This classification is based on the possible existence of multiple copies of a data item, also considering access rights of those copies. The complexity of coherence maintenance is strongly dependent on the introduced classes. In order to explore the properties of application behavior, including typical read/write patterns, while keeping the acceptable complexity of the algorithm, many solutions were proposed, among which h4RSW algorithms represent the majority.

Parameters closely retied to the DSM algorithm

a) Responsibility for the DSM management (centralized, distributed &wed, distributed/dynamic)

b) Consistency model (strict, sequential, processor, weak ,release, lazy release, entry, etc.)

c) Coherence policy (write-invalidate, write-update, type specific, etc.) Responsibility for DSM management can be centralized or distributed, and it determines which site has to handle actions related to the consistency maintenance in the system. Centralized management is easier to implement, but suffers from the lack of fault tolerance, while the distributed management can be defined statically or dynamically, eliminating bottlenecks and providing scalability. Distribution of responsibility for DSM management is closely related to the distribution of directory information, that can be organized in the form of linked lists or trees. Memory consistency model defines the legal ordering of memory references issued by some processor and observed by other processors. Stronger forms of consistency typically increase the memory access latency and the bandwidth requirements, and simplify the programming. More relaxed models result in better performance, at the expense of a higher involvement of the programmer in synchronizing the accesses to shared data. In strive to achieve an optimal behavior, systems with multiple consistency models adaptively applied to appropriate data types have been recently proposed. Coherence policy determines whether the existing copies of the data item being written to at one site will be immediately updated, or just invalidated on the other sites. The choice of coherence policy is related to the granularity of shared data. For very fine grain data items, the cost of update message is approximately the same as the cost of invalidation message.

Therefore, update policy is typical for systems with word-based coherence maintenance, and invalidation is used in coarse-grain systems. The efficiency of an invalidation approach is increased when the sequences of read and write to the same data item by various processors are not highly interleaved.

2. DSM IMPLEMENTATION:-

2.1. Hardware Level DSM Implementations:-

Hardware implementations of the DSM concept usually extend the principles found in traditional cache coherence schemes of scalable shared-memory architectures, Therefore, the unit of sharing is smaller-typically cache line size. Communication latency is considerably reduced, based on the advantage of megraing sharing, that also minimizes the effects of false sharing and thrashing. Searching and directory functions are much faster, compared to the software level implementations, as well. According to the general properties of memory system architecture, three groups of hardware DSM systems are regarded as especially interesting:

- CC-NUMA (Cache Coherent Non-Uniform Memory Architecture)
- COMA (Cache-Only Memory Architecture)
- RMS (Reflective Memory System architecture)

In a CC-NUMA system, the shared virtual address space is statically distributed across local memory modules of clusters. It is accessible by the local processors and by processors from other clusters, with quite different access latencies. The DSM mechanism relies on directories with organizations varying from a full-map storage to different dynamic organizations, such as single or double linked lists and fat trees. In order to minimize latency, static partitioning of data should be done with extreme care, in order to maximize the frequency of local access. The invalidation mechanism is typically applied, while some relaxed memory consistency model can be used as a source of performance improvement. Typical representatives of this type of DSM approach are Dash and SCI.

COMA architecture provides the dynamic partitioning of data in the form of distributed memories, organized as large second-level caches (attraction memories). There is no physical memory home location predetermined for particular data item, which can be simultaneously replicated in multiple caches. The existing COMA architectures are characterized by hierarchical network topologies that simplify two main problems in this

type of systems: finding an item and replacement of a cache block. In COMA architectures, the distribution of data across attraction memories is dynamically adaptable to the application behavior, therefore, they are less sensitive to static distribution of data than the NUMA architectures. Increased storage overhead for keeping the information typical for cache memory is inherent to the COMA architecture. However, some findings pointed out that this approach means an acceptably low amount of the overall system memory. Two most relevant representatives of COMA systems are DDM and KSR 1.Reflective memory systems are characterized with hardware- implemented update mechanism on the low level of data granularity. Some parts of local memory in each cluster can be declared as shared, and appropriately mapped into the common virtual space. Coherence maintenance of shared regions in these systems is based on full-replication algorithm. Following the assumption that all data written will be soon read by other sharing processors, those systems immediately propagates very change of all sharing sites, using a broadcast or multicast mechanism. Because of the property of "reflection," this kind of memory is also called "mirror memory." It results in high cost of write operations, especially when multiple writes to the same location occur, consequently, this architecture is the most convenient for the applications with a lower write frequency.

2.2. Software level DSM Implementations:-

The basic principle of the first software-implemented DSM mechanisms was quite alike to that of the virtual memory management, except that on page fault the data are supplied from local memory of the remote cluster instead from the local secondary storage. Software implementations of the DSM concept are usually built as a separate layer on the top of message passing model. According to the implementation level, several types of software-oriented DSM approaches can be recognized

1. Compiler implementations. In the cases where the DSM paradigm is applied at the level of parallel programming language, the shared address space is usually structured into logical units of sharing Therefore, shared data have to be declared as a specific type in the source program. In this approach, a can accesses to shared data are automatically converted into synchronization and coherence primitives. Language implementation can be portable between various systems, and recompiled if appropriate run-time primitives are available. Linda is an example of this software approach to DSM implementation

2. User-level runtime packages. The DSM mechanism is implemented by virtue of the run-time library routines, which are to be linked with an application program that uses the shared virtual address space. This approach is not only convenient for experimenting, but also flexible and easy to implement. IVY and Mermaid systems are based on run-time library routines

3.1. Operating system level &side the kernel). The interaction of scheduling, interrupt processing, and the application behavior can be efficiently examined if the DSM model is incorporated into the operating system kernel. The advantage of this approach is that the semantics of the underlying OS architecture can be preserved; hence, the applications can be ported from the local environment to the distributed system without being recompiled. Mirage is an example of system built according to this method.

3.2. Operating system level (outside the kernel). The DSM mechanism is incorporated in the specialized software controller, that can be (with minor modifications) used by different kernels. The same DSM mechanism can be used both by user and by the operating system kernel objects. One of the systems that follow this implementation level is Clouds.

The above classification should be taken conditionally, since all programming language implementations require some operating system support. Also, some programmers hints, at the level of the language, can help the run-time implementation to become more efficient. Software implementations are clearly inferior in performance to hardware implementations, but they are less expensive, can be suitable for a variety of underlying architectures, and can better take the advantage of the application characteristics. Problem-oriented shared memory, DSM in heterogeneous environments, and various sophisticated consistency mechanisms are mostly implemented in software.

2.3. Hybrid DSM Implementations:-

The integration of software and hardware methods, competitive management of DSM, and various consistency models, seem to be the most promising approach in the future of DSM. The idea to implement a combination of hardware and soft& ware is explored in the efforts to achieve scalability, limited in directory based schemes, that use the full-map hardware directories. Based on the observation that only a few simultaneously shared copies of the same shared data exist on average, the solution was found in which only a limited number of pointers for each directory entry are implemented in hardware. When more directory

storage is needed, it is managed by software. This principle is implemented in the MIT Alewife system. In order to gain better performance of DSM systems, researchers experiment lately with the use of multiple protocols within the same system, and even integrate message passing with the DSM mechanism. Innovative consistency models are also

being implemented, requiring additional activities of the programmer to suit the needs of application. In order to handle complexity of those, basically software solutions, special programmable protocol processors are added to the system, as it was done in the Stanford FLASH system.

3. CONCLUSION

The intention of this survey was to provide an extensive coverage of a11 relevant topics in an increasingly important area -distributed shared memory computing. A special attempt has been made to give the broadest overview of the proposed and existing approaches, in a uniform organizational manner. Because of the combined advantages of the shared memory and distributed systems, DSM solutions appear to be the most appropriate way toward large-scale high-performance systems with a reduced cost of parallel software development. In spite of that, building of successful commercial systems that follows the DSM paradigm is still in its infancy; consequently experimental and research efforts prevail. Therefore, the DSM field remains a very active research area. Some of the promising research typical applications and system configurations, synergistic combining of hardware and software implementations of the DSM concept, integrating of the shared memory and message passing programming paradigms, innovative system architecture (especially memory system), comparative performance evaluation, etc. From this point of view, tirther investment in exploring, developing, and implementing DSM systems seems to be quite justified and promising.

REFERENCE:-

1. http://en.wikipedia.org/wiki/Distributed_shared_memory

2. http://books.google.co.in/books/about/Distributed_Shared_Memory.html?id=YgXP-sumHu4C&redir_esc=y

3. http://www.studymode.com/

4.[Bershad1993] - Bemhad, B., N, Zekauskas M., J., Sawdon, W., A, "The Midway Diszibuted Shared Memory System," COMPCON 93. February 1993,

5. [Li1989,] - Li, K. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. PhD thesis, Department of Computer Science, Yale University, September 1986.

6. [Tanenbaum1992] - Bal, H., E., Tanenbaum, A, S., "Distributed programming with shared data," International Conjerance on Computer Languages '88, October 1988.

7.[Kulkarni *et al.* 1993] - Kulkarni, D.~C. et~al. Structuring Distributed Shared Memory with the "pi" Architecture. In *Proc. of the 13th Int'l Conf. on Distributed Computing Systems (ICDCS-13)*.

8.[AHUJA86] - Ahuja, S., Carriero, N., Gelernter, D.,"Linda and Friends," IEEE Computer, Vol. 19, No. 8. May 1986,

9.[AGARW90] - Agarwal, A., Lim, B., Kranq D., Kubiatowin, J., "APRIL: A Processor Architecture for Multiprocessing," Proceedings of the 17th Annual International-Symposium on Com$e;. Architecture, 1990,