CHAPTER 2  REVIEW OF LITERATURE

2.1 Overview

In this chapter the review of distributed system architecture model presented with their major advantages and disadvantages. In the next section, the literature review of distributed database implementation categories, DSM is implementing parameters for distributed shared memory configuration presented and then conventional DSM architecture strategies for distributed shared have been described.

2.2 Literature Review on Distributed Architecture

In a distributed computing system, the data are stored on numerous computers. The computers in a distributed computing system communicate with each other through different communication media such as the internet or high speed private networks. It can share disks, memory, or process contents. The site of such shared memory system may differ in size and task, extending from single computer to workstations. Each single machine can be identified by a number of different names, such as hosts, sites, nodes depend on which context they are projected. Here, the mainly term site is used to specify physical distribution of these computers. The common view of a distributed shared memory architecture depicted in Fig. 2.1. The highest variance amongst distributed data items and shared nothing parallel systems are that distributed data are typically geologically separated, are distinctly controlled and have a slower interconnection between sites (3). Another major difference is that in distributed system differentiation between global and local memory operation. A local memory operation is that in which process access and update shared data contents located at private memory of requesting site. While in global operation in that in which the requesting site will access and update data located at local memory on remote site (4). both kinds of execution require to be considered while programming the distributed shared memory mechanism, but after fetching specific data content process will perform operation similar way.
There are numerous explanations for implementing distributed shared memory (DSM) architecture comprising sharing of data items, availability and autonomy (3).

Sharing of data: The main benefit for the construction a distributed shared memory system is the facility of a background where users at one site may be able to access the data located on other sites. For example, in a distributed university system where each campus stores data related to own campus, it is possible for a user in one campus to access data in other campus. Without this ability, the transfer of student records from one campus to another campus would have to possibility to some external mechanism that would couple present systems.

Autonomy: The main benefit of sharing data by means of data distribution is that each site is able to retain a level of control over which data are stored in local memory. In centralized architecture, the overall database administrator will control shared data of the whole system from one site. In a distributed system, the overall shared data are controlled by the system database administrator. To, divide tasks there can be local administrator who is responsible for local transaction which works in cooperation with overall database administer while transaction is global. According to different
database structure every database administrator having different degree of autonomy. Sometimes local autonomy may have major benefits over system design while shared data contents are geographically more separated.

**Availability:** It is properties of distributed architecture to deal against failure in that if a specific site fails during transaction, then remaining sites of the system will continue operating. In contrast, if the shared data contents will be replicated to more than one site then, requesting site may fetch data form another site and continue to remain functional. So, failure of one site does not affect the functionality of the overall system or stop execution. A failure of a specific site is identified by system and appropriate recovering methodology is initiated.

The facilities on failed site not utilized by the system until the recovery action completed. After repair or recovery mechanism is completed, then failed come back into the system and participate in efficient executions. In centralized system recovery mechanism simple to implement, but, in case of distributed computing systems it is very complicated, the capability of most of the system to remain operational excluding the failure of one site results in increased availability. It is very critical for shared data systems used for real time submissions. For example, an airline system may have loss of ticket buyers again its competitors because of data access loss of system.

### 2.3 Distributed Architectural Models

The model defines that how the different components are configured in distributed architectures, according this property there are several models available. Amongst the most prominent ones are described below. In subsequent Fig. 2.2 to 2.4, P denotes processor, M denotes memory and disk are depicted by using cylinders. Memory, disk and topology can be configured according to system application requirement (3,5).

#### 2.3.1 Single Shared Memory Model

In this shared memory model different processors will access shared memory region by bus or high speed network. The main benefit of shared memory model is that in this kind of architecture processors can access shared
content without transfer to that requesting site and inter process communication will remain very efficient with software as shown Fig. 2.2. A process can send messages to other process much faster by using memory writes against direct a message by a communication method. It cannot be scalable beyond few numbers of processors because of bottleneck of common bus or interconnection network that is the main drawback of this architecture. (6). This kind of architecture does not accumulate more processes because if more number of processes means it is required to wait for turn to access memory. So, in this case process require to wait most of the time to get access to the memory region.

![Figure 2.2 Single Shared Memory Model (3)](image)

This shared memory structural design usually has bigger memory stores on each processor, so that addressing of the shared memory is escaped whenever possible. Always, some of the shared data are not cached and access will go to the shared memory region. The cached data also required to manage coherence view. If the shared data are updated or accessed in local memory by any processor than a cache by other processors will be updated or removed (3). While an increasing number of processor cache coherency will become overhead. Also shared memory computer are not scalable beyond some limit and cannot accommodate than a fewer number of processors.

### 2.3.2 Shared Disk Model

In the shared disk model each processor has its own local memories and all the processors can access each disk via bus of interconnection network as
shown in Fig. 2.3. There are two advantages of shared disk architecture over shared memory system. First, it is a less expensive architecture which provide a good level of fault tolerance. Second is each process having its local memory due to that memory bus will not become bottlenecks. If an individual processor or process fail or local memories fails than other processors will take over the task because shared data are located on shared disks. It is also possible to create structure fault tolerance with RAID architecture and it is found suitable in many applications (3).

Figure 2.3 Shared Disk Model (3)

Again, scalability is main difficulty is shared disk architectures. Since disk access now become a bottleneck against the memory bus bottleneck which is expensive in comparison of memory access. So, when there is the frequent number of access to disk while interconnection network to disk become bottlenecks (7). Compared to shared memory systems, shared disk models can scale to a somewhat greater number of processors, but communication across processors is slower, since it has to go through a communication network.

2.3.3 Shared Nothing Model

In a shared nothing model each node of the machine consists of a processor, memory and one or more disks. The processors on one node may communicate with another processor at another node with a high speed interconnection network shown in Fig. 2.4. A specific site facility as a server for data on disk that every site or node can retain. Since local request goes through the local disk of each processor. This architecture overcomes the
shortcomings of all requested I/O. Only process or processors can access remote data contents and results communication through the interconnected network (3,8). Moreover, a network of shared nothing architecture is designed for extension. So, network’s transmission capacity increased against more a number of nodes are added to the system. Therefore, this model is more scalable as a large number of processors are easily added (9).

![Figure 2.4 Shared Nothing Model (3)](image)

The main disadvantage of this model is communication overheads and remote disk access, which are greater than in a shared memory or shared disk model. In shared memory or shared disk architecture read/write to shared data involves software interaction at both ends.

### 2.3.4 Hierarchical Memory Model

The memory model combines the features of shared disk, shared memory, and shared nothing models. At the upper level, it contains high speed interconnection network which does not share memory or disk with each other. (3,10). Thus, the top layer is a shared nothing architecture. Every site of system will be the single shared memory network as depicted in Fig. 2.5.

On the other hand, each node could be a shared disk structure, and each of the systems sharing a set of disks could be a shared memory structure. Thus, a system could be constructed as a hierarchy, with a shared memory archetypal with a few processors at the base, and a shared nothing archetypal at the top, with possibly a shared disk construction in the middle (11). Fig. 2.5
illustrates a hierarchical model with shared memory nodes interconnected together in a shared nothing architecture. Commercial parallel systems today run on several of these architectures.

**Figure 2.5 Hierarchical Memory Model (3)**

An effort to decrease the complexity of programming such systems has yielded distributed virtual memory architectures, where logically there is a single shared memory, but physically there are multiple disjoint memory systems; the virtual memory mapping hardware, coupled with system software, allows each processor to view the disjoint memories as a single virtual memory (12). Since access speeds differ, depending on whether the data is available locally or not, such an architecture is also referred to as a non uniform memory architecture (NUMA).

### 2.3.5 Distributed Interconnection Networks

The distributed systems involve various components like memory, processors and disks which can interact with one another each other via a high speed communication network. Interconnection networks also called multistage interconnection networks or MINs, are high speed computer networks (3,13). They are connections between sites where each node can be a single processor or a group of processors or memory modules. Fig. 2.6 shows commonly used types of distributed interconnection networks.

These connections, transport or carry data from one processor to another or from the processor to the memory so that the task is broken down and computed in parallel. So, for example, you can have one network that is connected to a group of processors or sites at one end and a memory pool at the other end. The design in which the nodes are connected to each other is
known as topology (14). The two main types of topology are static and dynamic. Still sites are connected with different fashion through the network, according to that various topologies are defined as shown in Fig. 2.6.

![Figure 2.6 Distributed Interconnection Networks (1)](image)

### Figure 2.6 Distributed Interconnection Networks (1)

#### 2.4 Issues of Implementing DSM

A distributed shared memory structure consuming number of schemes and also having sub classes that is required to be considered. The scheme to which the DSM based system is programmed is most significant decision to build shared address space because its affect expense, software development and performance. To gain effective expense, programming and scalability are major points for each scheme. (15). An index mechanism required to execute on each access to find requested shared data is in local memory or not. If shared data is not local then the controller will retrieve shared data from the local memory of another site. The structure is also required to execute an action to maintain a coherence view of the system. Both index and action can be executed in both hardware, software and the combination of both (16). By this property system fall into three different categories. The selection of the specific category depends upon project goals and price, performance tradeoffs. Several aspect needs to be considered
while projecting shared memory mechanism as depicted in Table 2.1. Anyway, each of the categories can use the hardware as a part of system design.

**Table 2.1 Memory Design Parameters**

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Migration based</th>
<th>Replication based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client Server</td>
<td>Migration</td>
</tr>
<tr>
<td></td>
<td>Migration</td>
<td>Replication</td>
</tr>
<tr>
<td>Protocol</td>
<td>Write Invalidate</td>
<td>Write Update</td>
</tr>
<tr>
<td>Memory Access by</td>
<td>Software</td>
<td>Hardware</td>
</tr>
<tr>
<td>Consistency Model</td>
<td>Strict</td>
<td>Casual</td>
</tr>
<tr>
<td></td>
<td>Processor</td>
<td>Weak or other</td>
</tr>
<tr>
<td>Naming Scheme</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantics of Concurrent Access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Granularity</td>
<td>Object based</td>
<td>Shared variable based</td>
</tr>
<tr>
<td>Environment Support</td>
<td>Homogeneous</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>Implementation Level</td>
<td>Hardware</td>
<td>Software</td>
</tr>
<tr>
<td>Scaling Requirement</td>
<td>Object oriented</td>
<td>Procedure oriented</td>
</tr>
<tr>
<td>Design Criteria</td>
<td>Reliability</td>
<td></td>
</tr>
<tr>
<td>System Upgradation</td>
<td>Semantics of Concurrent Access</td>
<td></td>
</tr>
</tbody>
</table>
distributed shared memory studied and developed in past decades in various research projects explained in chapter 4 which fulfill their necessity and understanding. In their project district categories and memory, designing constraints to be considered to focus on specific goals.

2.4.1 DSM Algorithms

Conceptually, shared memory algorithms incorporate the local memory of computer within several distributed sites to provide the illusion to global address space and enable transparent virtual memory mechanism for its users. Based on data replication and migration basically four shared memory algorithms are studied in the literature and many variations are also available to form shared memory based on these algorithms (21-24). An implementation of this shared memory algorithm actually suffers with two complexities. One is static and dynamic distribution of each data amongst all sites that make shared memory possible. Second is to maintain a coherence view of shared data in the system.

As illustrated in Fig. 2.7, migration and replication are two frequent strategies to design shared memory (24). In replication multiple copies of the same data are stored in local memories of each node or copy of the requested data are replicated to requesting node. In migration scheme only a single copy of shared data available inside the system which are shipped to the requesting site. Based on these data, distributing strategies, DSM algorithms are classified into four types as shown in Fig. 2.8 (24-25). So, to minimize coherence overhead developer require to select specific data handling scheme and pattern of shared data management.
The consistency model of shared memory multiprocessor specifies conditions that how memory system available to the programmer or user and condition of memory access. It removes the confusion of characteristic expected by the programmer and actual condition is supported in system structure (26). It can be seen as an agreement between software and memory. This model describes the direction in which memory operations executed by different processors. It is significant to develop of parallel programs and formulate an integral part of the entire system design, including the architecture, the compiler, and the programming language (27). This condition required to be defined at each level where there an interface between user and system. At an architecture level, it touches the both hardware design expert and programmer who develop the software (28-29).

**Strict Consistency:** The most difficult consistency mechanism called strict consistency which is specified by condition: “Any read to a memory location x returns the value stored by the most recent write operation to x.” To explore this consistency mechanism more consider an example of Fig. 2.9 where two different processes P1 and P2. The execution done by each process described against time horizontally.

\[
\begin{align*}
P1 : & \quad W (X) 1 \\
P2 : & \quad R (X) 1
\end{align*}
\]
The horizontal line divides execution of each process. The notation \( W(X) \) 1 and \( R(X) \) 1 describes that a write to \( X \) with the value 1 and 1 read from \( X \) have been done, independently. Process \( P1 \) write the value 1 to location \( X \) which is read by process \( P2 \) subsequently. This characteristic best situation for strict consistency model.

**Sequential Consistency:** It is the most natural model greatly restricts the use of many performance optimizations commonly used by uniprocessor hardware and compiler designers, thereby reducing the benefit of using a multiprocessor (30). A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the 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 as described in Fig. 2.10. One of two aspect of this consistency model is to protect execution order amongst individual process execution. And the second is to maintain single execution direction amongst operation by all processors. The former concept is to make each operation executes atomically with respect to other memory operations.

\[
\begin{align*}
\text{Initially } & \text{FlagX} = \text{FlagY} = 0 \\
\text{FlagX} & = 1 \quad \text{FlagY} = 1 \\
\text{if (FlagY == 0)} & \quad \text{if (FlagX == 0)} \\
\text{critical section} & \quad \text{critical section}
\end{align*}
\]

**Figure 2.10 Sequential Consistency Memory**

**Causal Consistency:** The causal consistency model maintains the weakness of sequential consistency. It creates a partition between actions which are significantly casually related and those which are not. The shared memory to be appear as casually consistent. It is necessary that the memory, observe the following conditions: “Writes that are potentially causally related must be seen by all processes in the same order”. Parallel writes to memory may be considered in a different order on different processes (31). Consider ordering
of Fig. 2.11 which permitted in the casual consistency model, but which are not allowed in a sequentially consistent memory or a strictly consistent memory. Here, memory writes W(X)2 and W(X)3 are concurrent, so it is not required that all processes see them in the same order. If an application failure occurs when district processes view simultaneous events in different sequence, the system will violate the agreement done by casual consistent model.

![Figure 2.11 Sequence of Events for Causal Consistent Memory](image)

**Figure 2.11 Sequence of Events for Causal Consistent Memory**

**PRAM Consistency and Processor Consistency:** It is defined by the condition: “Writes done by a single process are received by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes”. The PRAM consistency model is easy to develop. It specifies that that there are no guarantees about the order in which district processes see write execution, except two or more writes from a single process must attain in order as though they were in a pipeline (32).

![Figure 2.12 Sequence of Events for PRAM Consistency](image)

**Figure 2.12 Sequence of Events for PRAM Consistency**
In this consistency mechanism all writes executed by different processes are concurrent. Above Fig. 2.12 showing sequences of events for PRAM consistency. Processor consistency is near enough to PRAM consistency (28). It is also well known as Goodman’s consistency model.

**Weak Consistency:** This consistency model is defined by stating the subsequent behavior that accesses to synchronization variables are sequentially consistent, no access to a synchronization variable is allowed to be executed until all preceding writes have completed everywhere and no data access is allowed to be executed until all preceding accesses to synchronization variables have been executed. Fig. 2.13 describes the sequence of events for weak consistent memory mechanism.

![Figure 2.13 Order of Events for Weak Consistency](image)

**Release Consistency:** This model offers access to memory system that’s about to enter in a critical region. Release consistency explained that critical region in just reached. These kind of accesses can be developed either as distinct executions or as ordinary execution on special variables. In another way the developer is responsible to program such way that specifies when to trigger them (7,33). For example, by execution library processes such as obtain, release or procedures such as enter critical region and leave critical region. A release consistency model having two characteristic that is different based on the execution orders they maintain among special operations. One is to preserve release consistency among special operation while the second is to preserve processor consistency among such operations.

**2.4.3 Environment Support**

An operating environment can be architected using different operating systems and topologies. It may be homogeneous or heterogeneous
depending upon its configuration. A network that uses a single network architecture and operating system is homogeneous. Homogeneous networks are the simplest networks to administer, but most corporate networks are heterogeneous networks (29,34). A heterogeneous network is a network connecting computers and other devices with different operating systems and/or protocols. For example, local area networks (LANs) that connects Microsoft windows and Linux based personal computers with apple Macintosh computers are heterogeneous. A heterogeneous network is the network of computer and other devices connected with each other across different operating systems and protocols.

2.4.4 Global Data Naming

When the system process wants to access the remote data than it has to identify the name and location of shared data and retrieve from there. Therefore, at least all shared data should be visible to all the sites. It is requires, some naming scheme to uniquely identify data contents of the shared region to avoid conflicts (15,28). One possible solution is the unique global name of each shared data contents in the virtual memory region. So, process wishing to access data will use a unique name to identify data and same way corporation process will use a unique name to globally identify data. The virtual memory manager at specific node will perform address translation of data contents (29). It is not useful if the granularity of data unit is less than specific unit. In this case requesting process have knowledge about the remote location of shared data.

2.3.5 Implementation Levels

The selection of implementation level depend a number of goals projected by the developer. It is also depends upon price performance tradeoffs. Sometimes hardware based implementation, high performance, adds more complexity in design Although usually higher in performance, hardware implementations to add more complex which is afforded by large scale machines (29). So, hardware based implementation is selected when performance is more important than budget. While for small scale systems can be implemented with software based methods like networks of personal
computers, built on commodity microprocessors. For, the mid-scale structure, selection of hybrid is good like a cluster of computers. Low cost additional hardware, used with hybrid solutions.

2.4.6 Implementation Language

The program design languages include two categories of languages system programming and application programming language. In a homogeneous scheme, it is very easy to care one specific system language. However, the system program design languages used on the heterogeneous sites may be exact different (17). This indicates that multiple equal implementations of a heterogeneous single address space may have to be done in the different languages. However, application program design languages should not be affected by the system program design languages as long as a functionally equal application program design interface is supported on all the sites. If a common application program design language is available on all the sites, then the same program would be usable on the sites. Otherwise, equal implementations of an application would have to be written, which growing the burden of using a heterogeneous system (35). Up to now, many of software DSM systems are built on Unix portability platforms, such as Digital Unix, AIX, Solaris etc., which is restricted to scientific applications that inhabit smaller market. As the popularity of Windows NT, specifically, as the presence of windows 2000 which incorporates windows NT and windows 9x together, it is important to support single address space on windows 9x/NT created platforms and support equal program design languages on windows platforms (19).

2.4.7 Memory Access Semantics

In a distributed computing system effectively sharing of memory possible to improve performance and reduce cost while executing more memory concentrated task. At this point hardware support is essential for good memory sharing mechanism (24). It is required to decide by programmer that which components of the system perform a memory access from hardware, operating system and memory management unit etc. All above components
having its limitation and good functionality. Choice of specific semantic depends upon project goal and its environment. The performance calculated based on two models, the software only method that support from the operating system to access remote memory and the hardware intensive method that uses a specific network interface to spread the memory system.

2.4.8 The Coherence Protocols

The DSM protocol uses two methodologies. One is a directory based where a single directory retains track of the sharing status of memory contents. And second uses snooping where every cache block is supplemented by the sharing status of that block. All cache controllers monitor the shared bus so they can update the sharing status of the block of data, if necessary ¾ write invalidate, a processor gain exclusive access of a block before writing by invalidating all other copies ¾ write-update, when a processor writes, it updates other shared copies of that block of data (25).

2.4.9 Semantics of Concurrent Access

It is described by capability of shared data access by more than one site in the distributed atmosphere (distributed lock management) or by two different paths towards memory simultaneously (for load balance). It is very difficult to take decisions when two different sited try to update the shared data at the same time (19). So, concurrent access policy is requires to define while programing virtual shared memory. This is where a lock, semaphore or barrier is required to allow collaboration between the sites for data access. Therefore, concurrent access require a suitable mechanism to deal with shared memory.

2.4.10 Future Scaling and Up gradation

As the requisite of today’s era scheme should be scalable and upgrade according to developmental change. The system must the capability of a database, network, or process to handle an increasing amount of work or its possible to be distended to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources are added. By time changes it’s also required to upgrade or add new features to system software. The complexity, cost,
time, and resources required to upgrade software will depend on a variety of factors, including whether your site has been customized either via with changes to the core system, plugins, or by an incorporation with another system.

2.4.11 Comparative Study of DSM Constraints

From the usual methods of architecting DSM system like hardware as cache coherence configuration and network interfaces and based on application, software based distributed shared memory systems can be implemented in programming library, at the operating system level or at underlying memory architect can be used to achieve the goal with different software parameters. But advantage is that these kind of systems are more portable and easy updatable software approach to DSM system design. The DSM systems implementation of virtual shared memory model of independent physically distributed memory system. It also involves various other implementation choices depend upon its design requirements includes distributed shared memory algorithms, level of implementation like in hardware, software or

![Figure 2.14 Comparative Study of DSM Constraints](image-url)
hybrid, critical region semantics, semantics of replication level, naming, structure specification has to be used to access remotely located in shared data, system consistency model, replication location for optimization, the granularity of shared data, remote access and caching/replication control of hardware or software, distributed shared memory controlled by virtual memory management software, OS or language run time system etc. (19-21). If we consider these design parameters, then by using many ways it is possible to design a distributed shared memory use the system’s virtual address space. Fig. 2.14 showing comparison of different constraints and its weightage from a scale of 10 which provides different ways in DSM architecture (8). According to design selections and issues shared memory can be implemented. However the design choice may vary according to system design requirement and performance, cost and scaling parameters.

2.5 Convention DSM Architecture

Two kinds of fundamental interprocess communication model message passing and shared memory paradigm used in distributed systems. With developers point of view shared memory application is quite easy to program, but challenging to extend beyond limited sites. It is required for shared data to run in parallel system which require distributed shared memory mechanism to share to make data available to all sites.

![Figure 2.15 Single Shared Memory Configuration](image)

In single shared memory simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies.
Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors. Above Fig. 2.15 show the single shared memory configuration. The Shared memory systems use uniform memory access (UMA), non-uniform memory access (NUMA) and cache only memory architecture (COMA) (28-29). While designing single shared memory architectures it provides advantages and disadvantages as listed below.

**Advantages:**

- The global address space provides a user friendly programming perspective to memory.
- The communication between processors through shared memory.
- It is easy of programming and lower latency.
- It is a model for unprocessors, the small scale of MPs.

**Disadvantages:**

- There is access time degradation due to a single bus inside the system.
- There may be bottleneck of shared region.
- It faces the lack of scalability between memory and CPUs.
- Here, failure of specific node may huge impact on the system.
- The programmer responsible for synchronization of constructs that ensure correct access of global memory.

**Figure 2.16 Distributed Shared Address Space**
A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. Each system has a number of advantages and drawbacks. It may be tightly coupled system or loosely coupled system.

In a distributed computing system individual processors perform independently same in virtual address space at program layer. This means that processor A can directly write into local memory which process B own as a part of the shared memory by simply using construct at the program level. The dispute with shared memory computers is that many processes need fast access to memory and will likely cache memory which has many difficulties (24). This architecture of shared memory is incorporated in to a basic DSM system such as SMP systems, PVP systems. The conventional distributed shared memory system is described in Fig. 2.16. In this kind of environment wrapper to shared data utilized to create shared memory illusion and usually use simple read and write statements to access directly application data structures when communicating between processes. In this type of a system where assignments statements do mapping directly to transport data between the processes and memory segments, this can be a very effective mechanism for communication depicted in Fig. 2.17 (21,28).

![Figure 2.17 Memory Mapping to Local Memories (1)](image)
In this case, after following basic rules two process A and B can communicate simply by reading and writing to a permitted location. The latency of communication is less than the hardware latency of memory access on the system. The available bandwidth between process communications is programmed to be close to main memory of the system (19). This can program communication technique more efficient. Here, some advantages and disadvantages listed for such a kind of distributed architecture.

Advantages:

- It shields programmer from Send/Receive primitives.
- It exploit locality of reference when a block is moved.
- Single address space, simplifies passing-by-reference and passing complex data structures.
- It is providing large virtual memory space.
- This DSM method uses simpler software interfaces and cheaper off the shelf hardware. Hence, cheaper than dedicated multiprocessor systems.
- There is no memory access bottleneck as no single bus.
- This memory programs portable as they use common DSM programming interface.

Disadvantages:

- It requires programmers need to understand consistency models to write correct programs.
- This DSM implementations use asynchronous message passing and hence cannot be more efficient than message passing implementations.
- By yielding control to DSM manager software, programmers cannot use their own message passing solutions.

With the use of different constraints and methods number of distributed techniques has been studied in the literature which retain different project goals. Some frequent used DSM strategies for distributed architecture are mentioned as below.
1. Single Bus Multiprocessors
2. Switch Multiprocessors
3. Non uniform multiple access
4. Page centered memory
5. Shared variable centered memory
6. Object centered memory

This dispersed shared memory structure design contains of several aspects like memory actions, remote memory access, transmission medium, encapsulation and methods, transfer contents, linear and virtual address space, data migration and possible remote access need to be considered.

**Table 2.2 Conventional DSM Implementations (19)**

<table>
<thead>
<tr>
<th>Distributed Architectures</th>
<th>Single Bus System</th>
<th>Switched Multicomputer</th>
<th>NUMA</th>
<th>Page based System</th>
<th>Shared Variable based System</th>
<th>Object based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory actions?</td>
<td>R-W</td>
<td>R-W</td>
<td>R-W</td>
<td>R-W</td>
<td>R-W</td>
<td>General</td>
</tr>
<tr>
<td>Remote memory access?</td>
<td>MMU</td>
<td>MMU</td>
<td>MMU</td>
<td>OS</td>
<td>System</td>
<td>System</td>
</tr>
<tr>
<td>Transmission through?</td>
<td>Bus</td>
<td>Bus</td>
<td>Bus</td>
<td>Network</td>
<td>Network</td>
<td>Network</td>
</tr>
<tr>
<td>Encapsulation/ methods?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Transfer content?</td>
<td>Block</td>
<td>Block</td>
<td>Page</td>
<td>Page</td>
<td>Variables</td>
<td>Object</td>
</tr>
<tr>
<td>Unattached memory?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linear/virtual space?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Data migration done by</td>
<td>H/W</td>
<td>H/W</td>
<td>S/W</td>
<td>S/W</td>
<td>S/W</td>
<td>S/W</td>
</tr>
<tr>
<td>Access using hardware?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 2.2 depicts the comparison of various different factors in different architectures. Each architecture describes consuming its own advantages and disadvantages, so it is required to select the appropriate technique as per requirement of the recent trends and environment. Here, for data sharing, cost effectiveness, reliability and commercial acceptance distributed shared memory is a very important approach to design, distribute system due these benefits this technique is selected as a research topic. The implementation choice usually depends on requirements and price performance balances and its applications are portable because of common shared memory interface, but shortcomings such as consistency model, algorithm and protocol require to be understood before starting development (24,32). In chapter 3 of this thesis, we have presented different methods for software based shared memory implementation.