Unit 3

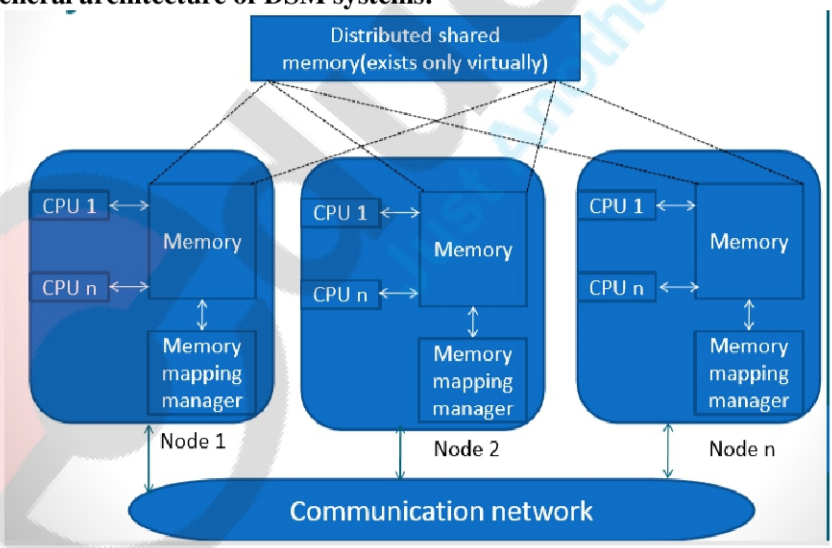
Distributed Shared Memory

Fundamental concepts of DSM, types of DSM, various hardware DSM systems, Consistency models, issues in designing and implementing DSM systems.

Introduction to DSM:

- The distributed shared memory (DSM) implements the shared memory model in distributed systems, which have no physical shared memory
- The shared memory model provides a virtual address space shared between all nodes
- Data moves between main memory and secondary memory (within a node) and between main memories of different nodes
- Each data object is owned by a node
  - Initial owner is the node that created object
  - Ownership can change as object moves from node to node
- When a process accesses data in the shared address space, the mapping manager maps shared memory address to physical memory (local or remote)

General architecture of DSM systems:



Design and Implementation Issues:

1. **Block Size:**

- As we know, transfer of the memory blocks is the major operation in the DSM systems. Therefore, block size matters a lot here. Block size is often referred to as the Granularity.
- Block size is the unit of sharing or unit of data transfer in the event of network block fault. Block size can be few words, pages or few pages. Size of the block depends on various factors like, paging overhead, thrashing, false sharing, and directory size.

2. Structure of Shared Memory Space:

- How the shared memory space is organized with data determines the structure of the shared memory space.
- It refers to the layout of shared data. It depends upon the type of application the DSM is going to handle.

3. Replacement Strategy:

- It may happen that one node might be accessing for a memory block from DSM when its own local memory is completely full. In such a case, when the memory block migrating from remote node reaches, it finds no space to get placed.
- Thus a replacement strategy of major concern in the design and implementation of the DSM systems. Certain block must be removed so as to place the new blocks in such a situation. Several techniques are used for the replacement of old blocks such as removal of Least Recently Used memory blocks.

4. Thrashing:

- Sometimes two or more processes might access the same memory block. Suppose two processes need to perform write operation on the same memory block.
- Since, to accomplish this, the block has to migrate in both directions at a very small interval of time, so it will be transferred back and forth at such a high rate that none of the two processes will be able to perform the operation accurately and completely. As such no real work is done. This condition is called thrashing.
- A technique should be incorporated, while designing the DSM systems to avoid thrashing.

5. Heterogeneity:

- DSM systems should be able to function on computers with different architectures.

Hardware based DSM:

- Uses hardware to eliminate software overhead
- May be hidden even from the operating system
- Usually provides sequential consistency
- May limit the size of the DSM system
- Examples of hardware based DSM systems are:
  - Shrimp
  - Memnet
  - DASH
  - Cray T3 Series

1. **Non Uniform Memory Access:**

- The Non Uniform Memory Access architecture, NUMA, approach is the simplest approach to hardware based distributed shared memory.
- Rather than maintaining a uniform access to memory, access time and memory behaviour differs, depending on the address that is accessed.
- NUMA machines are usually made up of several nodes with one or more CPUs and a block of memory each.
- Memory on the same node can then be accessed quickly, as the CPUs memory bus is connected directly to the local memory, while access to memory on a different node is routed via a scalable interconnect and is often not cached.
- The NUMA architectures utilize that local access, such as the executable and the stack of a process is fast, and that access to shared data can be fast also if it is placed in the processors local memory. Distributing data to nodes, so that most possible accesses are local, need to be considered by the programmer, or to a certain extent the compiler.

2. **Cache Coherent Non Uniform Memory Architecture:**

- CC-NUMA architectures are basically an approach to achieve the scalability of the NUMA architectures and maintain the programmability of UMA architectures.
- One or more processors are placed on the same node as a memory block and different nodes are then interconnected.
- CC-NUMA architectures do cache remote memory locations and keep them coherent, e.g. invalidate all cached copies on a write if a looser consistency model is not defined.
- This invalidation requires that the system keeps track of all cached copies of a cache line, and to maintain sequential consistency that all copies are invalidated before a write operation returns.

3. Cache Only Memory Architecture:

- A drastically different approach to maintaining cache coherence in distributed memory systems, is to eliminate the concept of main memory entirely. Instead of working with the concept of physical main memory, the system provides only cache memory.
- This cache memory is associated with a virtual memory space, rather than a physical memory, and thus the concept of main memory disappears. Instead memory blocks arealways tagged, and like the cache entries on an SMP system, the cache can be replicated and migrated freely among the nodes in the system

Consistency Models:

- A consistency model basically refers to the degree of consistency that has to be maintained for the shared-memory data for the memory to work correctly for a certain set of applications.
- It is defined as set of rules that applications must obey if they want the DSM system to provide degree of consistency guaranteed by the consistency model.

1. Strict:

Strict consistency is defined as:

- "Any read to a memory location x returns the value stored by the most recent write to x."
- That is, all writes instantaneously become visible to all processes.
- Implementation of strict consistency model requires the existence of an absolute global time so that memory read/write operations can be correctly ordered to make the meaning of "most recent" clear.
- However, absolute synchronization of clocks of all nodes of a distributed system is not possible.
- Therefore, existence of absolute global time in a distributed system is not possible.
- So, implementation of strict consistency model is practically impossible.

2. Sequential:

Proposed by Lamport.

- A shared memory is said to support the sequential consistency model if all processes see the same order of all memory access operations on the shared memory.
- The exact order in which the memory operations are interleaved does not matter.
- That is, if the three operations read(r1), write(w1), read(r2) are performed on a memory address in that order any of the ordering (r1, w1, r2), (r1, r2, w1), (w1, r1, r2), (w1, r2, r1), (r2, r1, w1), (r2, w1, r1) of the three operations is acceptable provided all processes see the same ordering.
- If one process sees one of the ordering of operations and another process sees a different one, the memory is not a sequentially consistent memory.
- A DSM system supporting sequential consistency model can be implemented by ensuring that no memory operation is started until all the previous ones have been completed.
- A sequentially consistent memory provides one-copy/singlecopy semantics because all the processes sharing a memory location always see exactly the same contents stored in it.

3. Causal:

- All processes see only those memory reference operation in the same(correct) order that are potentially causally related. Memory reference operations that are not potentially causally related may be seen by different processes in different orders.
- A memory reference operation(read/write) is said to be potentially causally related to another memory reference operation if the first one might have been influenced in any way by the second one.
- A shared memory system is said to support the causal consistency model if all write operations that are potentially causally related are seen by all processes in the same(correct) order.
- Write operations that are not potentially causally related may be seen by different processes in different orders.

4. Pipeline RAM:

- It ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline.

- Write operations performed by different processes may be seen by different processes in different order.
- For example, if w11 and w12 are two write operations performed by process p1 in that order and w21 and w22 are two write operations performed by process p2 in that order a process p3 may see them in order [(w11, w12), (w21, w22)] and another process p4 may see them in order [(w21, w22), (w11, w12)].
- It is simple and easy to implement and also has good performance.
- It can be implemented by simply sequencing the write operations performed at each node independently of the write operations performed on other nodes.
- In PRAM consistency all processes do not agree on the same order of memory reference operations.

5. Processor:

- It is very similar to PRAM consistency model with an additional restriction of memory coherence.
- That is processor consistent memory is both coherent and adheres to the PRAM consistency model.
- Memory coherence means that for any memory location all processes agree on the same order of all write operations to that location.
- Processor consistency ensures that all write operations performed on the same memory location are seen by all processes in the same order.
- Therefore, if w12 and w22 are write operations for writing to the same memory location x, all processes must see them in that order- w12 before w22 or w22 before w12.
- This means that both processes p3 and p4 must see the write operations in the same order, which may be either [(w11, w12), (w21, w22)] or [(w21, w22), (w11, w12)].

6. Weak:

- It takes advantage of the two characteristics common to many applications:
  I. It is not necessary to show the change in memory done by every write operations to other processes. The result of several write operations can be combined and sent to other processes only when they need it.
  II. Isolated accesses to shared variables are rare. That is, in many applications, a process makes several accesses to a set of shared variables and then no access at all to the variables in this set for long time.

- DSM system that supports weak consistency uses a special variable called synchronization variable. The operations on it are used to synchronize memory.
- For supporting weak consistency, the following requirements must be met:

i. Accesses to synchronization variables are sequentially consistent.
ii. No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
iii. No data access (read or write) is allowed to be performed until all previous accesses to synchronization variables have been performed.

7. Release:

- Weak consistency gets its performance from identifying synchronization variables, and thus isolating critical sections. What weak consistency does not do is to determine when a critical section is entered and when it is left.
- Because of this weak consistency must update both local writes and remote writes at each synchronization.
- If the type of synchronization could be determined, the system could reduce the update to the necessary type. This approach is called Release consistency
- Release consistency can be identified as:
    - Before an ordinary access to a shared variable is performed, all previous acquires done by the process must have completed successfully.
    - Before a release is allowed to be performed, all previous reads and writes done by the process must have completed.
    - The acquire and release accesses must be processor consistent.