© 2014 IJIRT | Volume 1 Issue 7 | ISSN: 2349-6002

PAGE BASED DISTRIBUTED SHARED MEMORY

Diksha Verma, Anjli Tyagi, Deepak Sharma
Department Of Information Technology, Dronacharya College Of Engineering

Abstract- **The distributed shared-memory programming paradigm has been receiving rising attention. Recent developments have resulted in viable distributed shared memory languages that are gaining vendors support, and several early compilers have been developed. This programming model has the potential of achieving a balance between ease-of-programming and performance. As in the shared-memory model, programmers need not to explicitly specify data accesses. Meanwhile, programmers can exploit data locality using a model that enables the placement of data close to the threads that process them, to reduce remote memory accesses. In computer architecture, distributed shared memory (DSM) 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).**

DSM system represents a successful hybrid of two parallel computer classes i.e. shared memory and distributed memory. It provides the shared memory abstraction in system with physically distributed memories and consequently combine the advantages of both approaches. DSM system is the memory system that is physically distributed but logically implements a single shared address space [5, 8, 11].

I. INTRODUCTION

The idea behind DSM is simple: Try to emulate the cache of a multiprocessor using the MMU and operating system software

In a DSM system, the address space is divided up into chunks, with the chunks being spread over all the processors in the system. When a processor references an address that is not local, a trap occurs, and the DSM software fetches the chunk containing the address and restarts the faulting instruction, which now completes successfully. 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.

II. REPLICATION

One improvement to the basic system that can improve performance considerably is to replicate chunks that are read only, read-only constants, or other read-only data structures. Another possibility is to replicate not only read-only chunks, but all chunks. As long as reads are being done, there is effectively no difference between replicating a read-only chunk and replicating a read-write chunk. However, if a replicated chunk is suddenly modified, inconsistent copies are in existence. The inconsistency is prevented by using some consistency protocols.

Finding the Owner

The simplest solution for finding the owner is by doing a broadcast, asking for the owner of the specified page to respond. An optimization is not just to ask who the owner is, but also to tell whether the sender wants to read or write and say whether it needs a copy of the page. The owner can then send a single message transferring ownership and the page as well, if needed. Broadcasting has the disadvantage of interrupting each processor, forcing it to inspect the request packet. For all the processors except the owner's, handling the interrupt is essentially wasted time. Broadcasting can use up considerable bandwidth, depending on the hardware.

There could be several other possibilities as well. In one of these, a process is designated as the page manager. It is the job of the manager to keep track of who owns each page. When a process, P, wants to read a page it does not have or wants to write a page it does not own, it sends a message to the page manager telling which operation it wants to perform and on which page. The manager then sends back a message telling the ownership, as required. A problem with this protocol is the potentially heavy load on the page manager, handling all the incoming requests. This problem can be reduced by having multiple page managers instead of just one.

Another possible algorithm is having each process keep track of the *probable* owner of each page. Requests for ownership are sent to the probable owner, which forwards them if ownership has changed. If ownership has changed several times, the request message will also have to be forwarded several times. At the start of execution and every *n* times ownership changes, the location of the new owner should be broadcast, to allow all processors to update their tables of probable owners.

Finding the Copies

Another important detail is how all the copies are found when they must be invalidated. Again, two possibilities present themselves. The first is to broadcast a message giving the page number and ask all processors holding the page to invalidate it. This approach works only if broadcast messages are totally reliable and can never be lost.

The second possibility is to have the owner or page manager maintain a list or copyset telling which processors hold which pages. When a page must be invalidated, the old owner, new owner, or page manager sends a message to each processor holding the page and waits for an acknowledgment. When each message has been acknowledged, the invalidation is complete.

Page Replacement

In a DSM system, as in any system using virtual memory, it can happen that a page is needed but that there is no free page frame in memory to hold it. When this situation occurs, a page must be evicted from memory to make room for the needed page. Two subproblems immediately arise: which page to evict and where to put it.

To a large extent, the choice of which page to evict can be made using traditional virtual memory algorithms, such as some approximation to the least recently used algorithm. As with conventional algorithms, it is worth keeping track of which pages are *'clean'* and which are *'dirty'*. In the context of DSM, a replicated page that another process owns is always a prime candidate to evict because it is known that another copy exists. Consequently, the page does not have to be saved anywhere. If a directory scheme is being used to keep track of copies, the owner or page manager must be informed of this decision, however. If pages are located by broadcasting, the page can just be discarded.

The second best choice is a replicated page that the evicting process owns. It is sufficient to pass ownership to one of the other copies but informing that process, the page manager, or both, depending on the implementation. The page itself need not be transferred, which results in a smaller message.

III. DATA GRANULARITY

Data granularity deals with amount of data process during the execution phase. It can be related to the amount of data exchanged between nodes at the end of execution phase as it is data that will be processed in the execution phase. In the framework the amount of data exchanged between nodes is usually a multiple of the physical page-size. The system using paging, regardless of the amount of sharing, the amount of data exchanged between nodes is usually a multiple of the physical page size of the basic architecture. Problem arises when system that consists very small data granularity are running on system that support very large physical pages. If the shared data is stored in contiguous memory location then most data can be stored in few physical pages. As a result the system performance degrades as the common physical page thrashes between different processors. To overcome with this difficulty the framework partitioned the shared data structure on to disjoint physical pages.

IV. CONSISTENCY PROBLEMS IN DISTRIBUTED SHARED MEMORY

To get acceptable performance from a Distributed Shared Memory System, data have to be placed near the processors who are using it. This is done by replicating and replacing data for read and write operations at a number of processors. Since several copies of data are stored in the local cache, read and write access can be performed efficiently. The caching technique increases the efficiency of Distributed Shared Memory Systems, but it also raises the consistency problems, which happens when a processor writes (modifies) the replicated shared data. How and when this change is visible by other processors who also have a copy of the shared data becomes an important issue. A memory is consistent if the value returned by a read operation is always the same as the value written by the most recent write operation to the same address [2]. In a distributed shared memory system, a processor has to access the shared virtual memory when page faults happen.

To reduce the communication cost initiated by this reason, it seems naturally to increase the page size. However, large page size produces the contention problem when a number of processes try to access the same page and it also triggers the false sharing problem, which, in turn, may increase the number of messages because of aggregation. [4].

V. CONCLUSION

As the framework uses memory page as the unit of data sharing, the entire address space (spread over all to the processor) is divided into pages. Whenever the virtual memory manager (VMM) finds a request to an address space which is not local, it asks the DSM manager to fetch that page from the remote machine. Such kind of page fault handling is simple and similar to what is done for local page faults. So for improving performance, the framework suggests for doing *replication* of the pages so that same page does not have to be transferred again and again. Keeping multiple copies of same page leads to the issue of consistency between these copies. This consistency issue is maintained by write-update coherence protocol. This is done by updating all processor about the write operation. The page manager maintains a list or copy set telling which processors hold which pages. When a page must be invalidated, the old owner, new owner, or page manager sends a message to each processor holding the page and waits for an acknowledgment. When each message has been acknowledged, the invalidation is complete. In this DSM framework, it may happen that a page is needed but that there is no free page frame in memory to hold it. When this situation occurs, a page must be ejected from memory to make room for the needed page. Two sub problems immediately arise: which page to eject and where to put it. In the context of framework, a replicated page that another process owns is always a prime candidate to eject because it is known that another copy exists.

REFERENCES

[1] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. ACM Transactions on Computer Systems, 7(4):321–359, November 1989.

[2] Changhun Lee "Distributed Shared Memory" Proceedings on the 15th CISL Winter Workshop Kushu, Japan, February 2002.

[3] Yvon Jégou "Implementation of Page Management in Mome, a User-Level DSM" INRIA November 2003.

[4] Bal Gopal, Pankaj Kumar “Perfect Memory Consistency Model for Improving Parallelism in DSM System” Proceedings of Third international conference on information processing (ICIP-09), 7-9 Aug 2009.

[5] Sarita V. Adve, Vijay S. Paiand Parthasarathy Ranganathan “Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems” Proceedings of the IEEE, VOL. 87, NO. 3, MARCH 1999, pp. 445-455.

[6] Bal Gopal and Pankaj Kumar “Framework for Improving Parallelism by Write-Update Coherence Protocol in Distributed Shared Memory System” IJRTE, Issue. 1, Vol. 2, June 2009 pp. 201-204