CHAPTER 7

PAGING AND PERFORMANCE ANALYSIS

The paging system of our proposed DSM framework closely follows the shared memory in multiprocessor sphere. The entire address space (spread over all the nodes) 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 [1, 6]. To improve performance, replication of the pages is done so that same page does not have to be transferred again and again. This especially improves performance for read only pages. Keeping multiple copies of same page leads the issues of consistency between these copies [3, 8, 13]. If pages are not replicated, achieving consistency is not difficult. But for the replicated copies, consistency maintenance becomes difficult and this is solved in the previous chapter.

The framework relies on the message passing technique across the network for data exchange between two processors. So the framework has to present a uniform global view of the entire address space (consisting of physical memories of all the processors in the network) to a program executing on any processor. A DSM manager on a particular processor
would capture all the remote data accesses made by any process running on that machine.

7.1 Granularity

Granularity refers to the size of a data unit in which it exists in the distributed shared memory. This is an important decision which essentially governs the design of a DSM. The immediate successor of shared memory from multiprocessor world would have a page as the unit for data transfer. Granularity describes the size of the minimum unit of shared memory [2, 7, 11]. In our DSM framework it is the page-size. The protocols and hardware that are used in framework to propagate updates will have an influence on the choice of granularity. In the framework the efficiency is maximized by making the granularity into a multiple of the size used by these. The framework is designed to optimize the passing of pages around the network so that the page-size will be determined.

7.1.1 Effect of 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 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.

7.1.2 Effect of Execution Granularity

Execution granularity is the amount of execution a process has to do between synchronization and communication point in a multiprocess computation. Execution Granularity Ratio (EGRatio) is calculated by time spent in the execution \( T_E \) and they spent in the requesting data for the execution \( T_R \) i.e.

\[
EGRatio = \frac{T_E}{T_R}
\]

The speedup of any implementation is good if it spends more time on execution rather than requesting for data for requesting. So for good speed \( T_E \) should be greater than \( T_R \). Or we can say that for good speed the EGRatio should be greater than 1 i.e.

\[
EGRatio > 1
\]

If EGRatio is less than 1 it means that system spent more time on requesting the data. Hence it will have slow speed. Figure 7.1 shows the plot for EGRatio. The shaded region shows the region where the EGRatio is greater than 1. The systems differ with each other in
size of data to be transferred, speed of communication medium and other overhead of transfer. So for a particular implementation the EGRatio of the systems should fall in the shaded region.

![Figure 7.1 EGRatio requirement.](image)

So for perfect EGRatio it is good to calculate the execution requirement for the framework by matching the minimum communication time for requiring data.

### 7.2 Thrashing

Thrashing occurs when there are several processes that have frequent needs for access to the same page, this means that ownership will be difficult to determine and the page will be transferred back and forth across the network at a very high frequency. These processes will spend much of their time waiting for the page to be returned to them and so their performance will suffer [2, 5].

The solution to thrashing it is to allow several copies of the page to exist simultaneously on a number of machines and to institute some means
of ensuring that they all display a consistent view of the data that they contain.

### 7.3 False Sharing

A process that accesses one variable is very likely to access other variables located near to it in memory. However this consideration is often overridden by the fact that a large page-size increases the possibility of false sharing. This occurs when a number of processes seek to access unrelated data items that happen to be located on the same page. False sharing particularly occurs if a page contains a key variable that is accessed very often, this will mean that process will experience considerable delays in accessing any other variables located on the same page [4, 9, 12]. This problem has been moderated somewhat for page-based systems by adopting a mode of operation where key variables are stored alone on one page, so that accessing them will not cause any difficulties in accessing other unrelated data items. For instance, two user-defined objects are allocated to the same page and two processors P1 and P2 are accessing the page concurrently. As a result, the page has to travel across the network each time P1 wants to write in it when it is being held by P2 or vice versa. This problem occurs mainly because of the difference between the programmer’s view of sharing and sharing at the DSM level and can lead to the thrashing problem.

False sharing is a particularly serious problem for DSM systems for two reasons:

1. The consistency units are large, so false sharing is very common and
2. The latencies associated with detecting modifications and communicating are large, so unnecessary faults and messages are particularly expensive.

To overcome with the problem of false sharing, the framework allows multiple processes to write concurrently into a shared page with the updates being merged at the appropriate synchronization point in accordance with the definition of processor consistency and write-update coherence protocol.

7.4 Paging Implementation

Paging implementation in our framework is an extension of traditional Virtual Memory systems. It is implemented in a way that it will be completely transparent with respect to the memory system, i.e. memory system is completely hidden from users of the system. It is difficult to choose the right size for the page and/or block because page and/or block not only depends on the system characteristics but also on applications.

7.5 Overhead Analysis

The overhead in proposed DSM framework is the large amount of communication that is required to maintain consistency. The network hardware for the DSM framework is optimized for passing pages across the network. It divided the address space into pages, each page being located in the local memory of one machine in the system. When a processor tries to access an address in a page that is not local the DSM
software identifies the page, locates it and fetches it from the machine it is currently stored on. Thus the virtual memory consists of pages, stored in local memory of a machine on the network, that migrate across the network as different processors need to access their contents. This migration is required for read access as well as writes accesses, because the only way data can migrate across the network is if the page it is located on is transferred to another machine.

We have categorized the times of DSM framework into three cases for the calculation of overhead.

1. Time took for the servicing of page faults.

2. Includes updating state information for shared segment and coherence maintenance.

3. Network communication between nodes and a page transfer.

### 7.5.1 Page fault Servicing Overhead

As the framework is used Multiple Read Multiple Write (MRMW) algorithm for the replication, it improves system throughput by having all processing nodes synchronously. The throughput of the framework depends on the following two factors:

1. Average overhead of parallel context switches between the processes.

2. Average fraction of page fault service time overhead.

The context switch occurs when the execution switches from one process to another process. As in the framework we have not
concentrated on process switch. If $T_{PF}$ is the time to service the local page fault and the $N_{PF}$ is the number of page fault then the total time overhead for servicing of page faults will be

$$T_{TPF} = N_{PF} \times T_{PF}$$

If $n$ is the number of shared nodes then average effective overhead of page faults will be

$$T_{EPF} = \frac{T_{TPF}}{n}$$

The average fraction of page fault service time overhead will be given by

$$F_{PF} = \frac{T_{EPF}}{T_{PF}}$$

Figure 7.2 Average fraction of page fault with number of page fault
Average fraction of page fault depends upon number of shared nodes and number of page fault. It is observed that average fraction of page fault increases when page fault increases and decreases when shared nodes increase. Analysis of average fraction of page fault with respect to number of page fault is shown in figure 7.2. Average fraction of page fault tells us about the average memory access in the framework. More average fraction of page faults creates more overhead in the framework.

7.5.2 Coherency Maintenance Overhead

The framework uses write-update coherence protocol for the consistency maintenance. The protocol supports multiple copies of data to be transferred to different nodes. So the total number of bits to be transferred will depend upon the number of processors, number of bits in page, and the number of cache in the framework. Let

\[ n = \text{number of processors} \]
\[ b = \text{number of words per page} \]
\[ w = \text{number of bits per word} \]
\[ m = \text{number of blocks in each cache of the B memory module} \]
\[ c = \text{number of caches in each of the P caches} \]

So the total number of bits in the system to transfer

\[ D = nbw(m + c) \]
If \( N_X \) is the number of bits dedicated to coherency function for maintaining the consistency and coherence protocol then the corresponding overhead will be

\[
O_X = \frac{N_X}{D}
\]

The coherency overhead increases as number of processor increases whereas the corresponding overhead will decrease.

### 7.5.3 Network Communication Overhead

The network communication overhead can be expressed as a sum of two types of cost. Fixed cost consists of the overhead associated with the virtual memory subsystem and the cost of sending a data request to the data server. Variable cost consists of cost of sending the data to requester. The overhead for the network communication can be calculated by

\[
NC_{Overhead} = FixedCost + VariableCost \quad Eq.(7.1)
\]

Fixed cost depends upon virtual memory overhead (VMO), data request cost (DRC) and the page-size (PS). The virtual memory cost and the data request depends upon the size of data to be transferred to other nodes. So the fixed cost can be calculated by

\[
FixedCost = \frac{VMO + DRC}{PS} \quad Eq.(7.2)
\]

It implies that if the framework acquires high virtual memory overhead and high data request cost then the fixed cost can be
minimized by increasing page-size. The variable cost controls the latency of data and depends upon server processing cost (SPC), page-size (PS) and media bandwidth (MB). It can be calculated by

\[
VariableCost = (SPC) \times PS + \frac{PS}{MB}
\]

Eq.(7.3)

It implies that for getting low latency the page-size should be kept small. For the proper implementation of distributed shared memory the data transfer cost should be minimum. If there are more data have to be transferred then the page-size should be kept small as the variable cost increases rapidly otherwise the page-size should be kept large. So for minimizing the overhead it is suggested to take variable page-size.

7.6 Page size calculation

The unit of data managed and transferred by DSM is a data block which is called DSM page. In the DSM framework we have taken heterogeneous page system which supports different virtual memory page-size, so choose a size for DSM page become an important issue. If we choose small virtual memory page-size for DSM page then multiple small virtual memory pages fit exactly in one DSM page, hence they can be treated as group of page-fault. If we choose large virtual memory page-size for DSM page then more data than necessary data may be moved between hosts. The one thing which also be considered that when data items in the same DSM page are being updated by multiple hosts at the same time, causing large number of page transfer between the hosts without much progress in
the execution of the application. This causes false sharing, where non overlapped regions in the same page are shared and updated by different host, causing repeated page transfer.

So for proper calculation of page-size we have concentrated on the network communication overhead. Therefore by the equation 7.1 we get network communication overhead (NCO)

\[
NCO = \frac{VMO + DRC}{PS} + (SPC \times PS + \frac{PS}{MB})
\]

Let \(VMO + DRC = 'A'\) and \(SPC + 1/MB = 'B'\) the network communication overhead will be

\[
NCO = \frac{A}{PS} + B.PS
\]

As the page-size is not taken fixed and is dependent on the overhead of the network communication therefore differentiating NCO with respect to PS we will get

\[
\frac{dNCO}{dPS} = -\frac{A}{PS^2} + B \quad \text{Eq.(7.4)}
\]

\(\frac{dNCO}{dPS}\) will give the network communication overhead with respect to the page-size. It tells the rate of change of network communication overhead with respect to page size.
**Figure 7.3** Graph for Equation (7.4)

As the network communication overhead with respect to the page-size is constant and fig. 7.3 shows that the function is continuous and reaches to maximum. So the equation 7.4 will be

\[ \theta = \frac{A}{PS^2} + B \]

i.e

\[ PS = \sqrt[3]{A} \]

Putting the value of ‘A’ and ‘B’

\[ PS = \frac{VMO + DRC}{(SPC + \frac{1}{MB})} \]

And finally

\[ PS = \sqrt{\frac{(VMO + DRC) \times MB}{(SPC \times MB) + 1}} \]  \hspace{1cm} \text{Eq.(7.5)}
By equation 7.5 we can say that the page-size of the framework depends upon virtual memory overhead (VMO), data request cost (DRC), server processing cost (SPC) and media bandwidth (MB). As the value SPC*MB is too high and adding 1 into it doesn’t affect so let (SPC*MB) = 1, then the page-size will be

\[ PS = \sqrt{\frac{VMO + DRC}{SPC}} \]  
Eq.(7.6)

Above page size doesn’t include number of reads/writes and number of shared nodes for writing updates. Considering these issues the page size with write update will be

\[ PS_{wu} = \frac{n_w}{r + w} \sqrt{\frac{VMO + DRC}{SPC}} \]  
Eq.(7.7)

n = number of shared nodes where write updates have to be done
w = number of writes
r = number of reads.

The validation of PSwu is done through the correlation method. The method tells that how page size is correlated with VMO, DRC and SPC. It tells the corresponding changes occur in page size with respect to other. The negative correlation coefficient (table A1.1) between SPC and PS shows that both factors will move in the opposite direction i.e. increment in SPC decreases the page size and vice-versa. The negative correlation coefficient (table A1.2) between VMO and PS shows that increment in VMO decreases the page size and vice-versa. The negative correlation coefficient between (table A1.3) DRC and PS shows that increment in
DRC decreases the page size and vice-versa. But the positive correlation coefficient (table A1.4) between VMO+DRC and PS shows that increment in VMO+DRC increases PS. So we can say that PS is proportional to VMO+DRC and inversely proportional to SPC, which is similar as Eq.(7.6).

7.7 Page size analysis

Page size with respect to nodes:

As compare to the page size without write updates (PS), the page size with writes updates (PSwu) is increasing when number of shared nodes is increasing in the framework. By the figure 7.4 we can say that the page size with updates (PSwu) is increased up to five times as compare to the page size without write updates (PS). The reason for high rise in PSwu due to shared nodes is that as number of shared node is increased number of address space will also be increased. Increment in number of shared nodes will also increase in pages.

Figure 7.4 Page Size analysis with respect to shared nodes
**Page size with respect to writes:**

As compare to the page size without write updates (PS), the page size with writes updates (PSwu) is increasing when number of writes is increasing in the framework. By the figure 7.5 we can say that the page size with updates (PSwu) is increased up to three times as compared to page size (PS) without write updates.

![Figure 7.5 Page Size analysis with respect to writes](image)

**Page size with respect to reads:**

As compare to the page sizes with shared nodes and writes, the page size with writes updates (PSwu) in reads is slowly increasing as number of reads increases in the framework. The figure 7.6 shows that the page size with write updates (PSwu) increases with little factor as compared to page size without write updates (PS). So we can say that the reads does not affect too much on the page size.
The effect of reads on the page size comparatively increases less with respect to shared nodes and writes. Figure 7.7 shows the page size analysis for shared nodes, reads and writes in combine. It shows that the page size with shared nodes increases more as compare to other.

As the page size not only depends upon these factors but also depends upon factors like VMO, DRC and SPC. The page size with write updates (PSwu) decreases as server processing cost increases. As both
VMO and DRC increases the page size with write updates (PSwu) also increases. So it will be a good strategy that when both VMO and DRC increase, SPC should also increases.

An analysis of page size with updates (PSwu) is shown in fig. 7.8. We can see that when SPC increases PSwu decreases and when VMO and DRC increases the page size with write updates also increases. As the number of writes and number of shared nodes increases, the network communication overhead also increases and as a result the page size of the system also increases.
REFERENCE


