<table>
<thead>
<tr>
<th>Title</th>
<th>Performance analysis of the doubly-linked list protocol family for distributed shared memory systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Lau, Albert CK; Yung, Nelson HC; Cheung, YS</td>
</tr>
<tr>
<td>Citation</td>
<td>Ieee International Conference On Algorithms And Architectures For Parallel Processing, 1996, p. 365-372</td>
</tr>
<tr>
<td>Issued Date</td>
<td>1996</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10722/45849">http://hdl.handle.net/10722/45849</a></td>
</tr>
<tr>
<td>Rights</td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.; ©1996 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.</td>
</tr>
</tbody>
</table>
ABSTRACT The doubly-linked list (DLL) protocol provides a memory efficient, scalable, high-performance and yet easy to implement method to maintain memory coherence in distributed shared memory (DSM) systems. In this paper, the performance analysis of the DLL family of protocols is presented. Theoretically, the DLL protocol with stable owners has the shortest remote memory access latency among the DLL protocol family. According to the simulated performance evaluation, the DLL-S protocol is 65.7% faster than the DDM algorithm for the linear equation solver, and is 16.5% faster for the matrix multiplier. From the trend of the performance figures, it is predicted that the improvement in performance due to the DLL-S protocol will be considerably greater when a larger number of processors are used, indicating that the DLL-S protocol is also the most scalable of the protocols tested.

1. Introduction

Distributed Shared-Memory (DSM) [1] is an important aspect of parallel processing because it allows programmers to use the shared-memory programming model on systems that have distributed main memory. Traditionally, interprocessor communications in distributed memory multiprocessors rely on message passing, in which the programmers are responsible for handling all the formatting, sending and receiving of messages. With DSM, however, interprocessor communications can be performed simply by reading and writing the shared memory space, while the underlying mechanism is transparent to the programmers. In order to create a shared memory space from physically distributed memory, a DSM protocol is generally required to handle the remote memory accesses and to maintain memory coherence.

In this study, the base system architecture of the DSM system is assumed to be a generalized multiprocessor model, called the hierarchical cluster model [2]. In this model, multiple clusters are connected by an interconnection network (Figure 1a). Each clusters has a small number of Processing Elements (PEs) and its own local memory (Figure 1b). In the hierarchical cluster model, programmers have to use both the shared-memory model for intra-cluster communications and the message-passing model for inter-cluster communications. With DSM, the complications of the underlying architecture are hidden from the programmers, who see only a uniform contiguous memory space.

One of the early software DSM system is IVY [3], which implemented the DSM concept as virtual shared memory. In IVY, when a page fault occurs in a cluster’s local memory, the faulting page is fetched from a remote cluster that has a valid copy of the page, instead of loading from disk. It experimented with various DSM algorithms and concluded that the Dynamic Distributed Manager (DDM) algorithm generally had the best performance. In the DDM algorithm, pages can migrate and replicate freely throughout the system as needed for shared accesses by different clusters. The page management is performed by individual owner cluster of a page that keeps the copy-set, which is the set of clusters that has valid copies of the page. Whenever there is a write access to a page, the owner of the page invalidates all other copies of the page in the system listed in the copy-set, then transfers the ownership to the cluster that writes to the page. As the DDM algorithm is an extension of the basic virtual memory system, a standard feature supported by virtually all contemporary microprocessors, the overhead caused by the algorithm is small. However, there are rooms for improvement in the DDM algorithm. First, the DDM algorithm performs write invalidation by using information from the copy-sets, which are dynamic memory structures whose maximum size is the number of clusters in the system. The worst case total size of the copy-sets is thus equal to the number of pages in the system times the maximum size of a copy-set — for a system with 1024 clusters and 128 Mbyte main memory with 1 Kbyte pages, the maximum total size of all copy-sets in a cluster has $128 \times 2^{10}$ (more than 128 millions) entries! This severely limits the scalability of the system. Second, the burst of invalidation messages generated by the owner during write-invalidations may congest the part of network around the cluster — in the worst case, for a system with $N$ clusters, if every cluster in the system is invalidating a page in every other clusters, the number of messages sent will be $N \times (N - 1)$, i.e., the maximum instantaneous number of invalidation message in the system is $O(N^2)$. In Li’s paper [3], a method to partially distribute the copy-set using trees of clusters was proposed; however, as dynamic memory structures are still needed in the algorithm, the problem is still not completely solved.

To address this problem, the Doubly-Linked List (DLL) protocol [4] was proposed. The DLL protocol is a software DSM algorithm that is suitable for implementation in the distributed operating systems of a wide varieties of multiprocessor systems. As in the DDM algorithm, the DLL protocol is transparent to programmers and allows migrations and replications of memory pages. However, instead of using copy-sets, linked lists of clusters formed by the P-links which require constant storage space in the page tables are used to perform write invalidation. The total space required to store all the P-links in a cluster is equal to the number of pages in the
system — for the same 1024 clusters system as mentioned above, only a constant 128-2^{10} (1024 times fewer than the worst case of DDM) entries is needed to store the P-links. Moreover, the use of links allows invalidations to be performed in a distributed way in which the owners need not send large bursts of invalidation messages — for a system with N clusters, the maximum instantaneous number of messages is 2N, i.e., O(N). Furthermore, in the DDM algorithm, every cluster performing an invalidation of a page needs to send an acknowledgment message, whereas in the DLL protocol, only one acknowledgment message is needed for the invalidation of a page. Therefore, in theory, the DLL protocol minimizes both the possibility of network congestion and the number of messages used.

In this paper, the performance analysis of the basic DLL protocol (DLL-B), the DLL protocol with N-link Reduction (DLL-R) and the DLL protocol with stable owners (DLL-S), is presented. In the basic protocol, the cluster that most recently acquires a page becomes the owner of the page. Although this method lengthens the time required to locate the owners, it speeds up the read-modify-write memory access sequences which are used in many applications. As the read operations change the owner of the page to the requesting cluster, it can then perform the write invalidation directly. The DLL-R protocol is developed to shorten the time required to locate the owners by partially reducing the length of the chains of N-links, and yet it preserves the quick read-modify-write advantage of the basic protocol. In the new DLL-S protocol, ownership is not transferred during read accesses, thus eliminating the need to trace through chains of N-links to locate the owner. In addition, multiple read accesses can be serviced simultaneously by clusters that have copies of the page. However, it loses the fast read-modify-write advantage of the basic protocol.

Theoretically, the DLL-S protocol has the shortest remote memory access latency among the DLL family of protocols. According to the simulation study, the DLL-S protocol achieves an improvement of 65.7% over the DDM algorithm for the linear equation solver, and an improvement of 16.5% for the matrix multiplier.

The organization of the paper is that the DLL-B and the DLL-R protocols are outlined in the section 2. The DLL-S protocol is discussed in section 3. A theoretically analysis of the protocols is presented in section 4. The performance evaluations of the protocols by simulations are presented in section 5, and finally, conclusions are made in section 6.

2. The Basic DLL Protocol

In the DLL protocol family, the memory space is divided into fixed size pages as in virtual memory systems (Figure 2). Each cluster maintains its own page table, which contains information about all the memory pages in the system. Each memory page in the page table can have one of the three states -- E (exclusive), S (shared) or I (invalid). E state means the cluster has the only copy of the page in the whole system. S state the cluster has a copy of the page but it is not the only copy. I state means the cluster does not have a valid copy of the page.

Every page has an owner cluster, although page ownership can be transferred. In the DLL-B protocol, the owner of a page is the cluster that most recently acquired the page. It is the responsibility of the owner to supply the page to the requesting clusters. Also contained in the page table are the P-link and the N-link for each page. The P-link points to the cluster that is the previous owner of the page, while the N-link points to the cluster to which the page ownership is given. A null N-link means the cluster is the owner of the page. From any cluster, the owner of a page can be reached by following its N-links; and from the owner of a page, all clusters in the system that have copies of a page can be visited by following the P-links.

2.1 Read Accesses

Read accesses to pages with E or S states are performed locally, however, when a cluster performs a read access to an I page, a remote read access is required to obtain the page from the owner. The cluster sends a read-request (RR) message to the page's N-link. Following the chain of N-links, the message will eventually reach the owner of the page, which replies by sending a copy of the requested page back to the requesting cluster through the read-data (RD) message. It then points its N-link to the requesting cluster and sets the page state to S. The requesting cluster, on receiving the RD message, copies the page to its local memory, sets the page state to S and its N-link to NULL and points its P-link to the servicing cluster. At this point, the requesting cluster becomes the new owner of the page and completes the read access.

The following is an example of a remote read access: Assume cluster c0 is the owner of page p0 and cluster c1, whose p0 is I state and the N-link of p0 points to c0, now performs a read access to p0. Therefore, c1 sends an RR message to c0, which replies by sending an RD message containing a copy of p0 to c1, changes the state of p0 to S and sets the N-link to point to c1. When c1 receives the RD message, it copies the page p0 to its local memory, changes the page state to S, N-link to NULL and P-link to c0, and completes the read access. The process is depicted in Figure 3.

2.2 Write Accesses

If a cluster performs a write access to an E page, the request can be completed locally; otherwise, a remote memory access is generated. If the page state is S, all other copies of the page in the system must be invalided to maintain memory coherence. A write-invalidate (WI) message is sent through the chain of N-links to the owner, which sets its own page state to I and sends a write-invalidate-forward (WIF) message to the page's P-link. The WIF message goes through the chain of P-links, thus invalidating all copies of the page in the system, except the one in the requesting cluster, which ignores the
message. When the WIF message reaches the end of the P-links chain, a write-invalidate-performed (WIP) message is sent back to the requesting cluster. On receiving the WIP message, the requesting cluster sets the page state to E, and both the N-link and the P-link to NULL. At this point, the write access is completed.

The following is an example of a write access to an S page. First, assume the page table state in Table 1 and now c0 performs a write access to p0. Since p0 is in state S in c0, other clusters with copies of p0 must have their copies invalidated. Therefore, a WI message is sent to the cluster pointed to by the N-link, i.e., cluster c1. Following the N-links, c1 forwards the WI message to c2, which is the current owner of p0. Cluster c2 then sends a WIF message to the cluster pointed to by its P-link, i.e., cluster c0, changes the state of its own p0 to I, and resets its P-link to null and N-link to c0. Cluster c1, on receiving the WIF message, changes the state of p0 to L, forwards the message to the cluster pointed to by its own P-link, i.e., cluster c0, and reset its P-N-links to null and c0, respectively. When c0 receives the WIF message, as it is the requesting cluster, it ignores the message. Since c0’s P-link is null, all copies of p0 in the system, except the one in c0, are invalidated. At this point, c0 should send a WIP message back to the requesting cluster; however, as the requesting cluster is c0 itself, so this message is skipped. Finally, c0 changes the state of p0 to E, sets both of its P and N-links to null, and completes the write access. The cluster c0 becomes the new exclusive owner of p0. The process is depicted in Figure 5.

2.3 Advantages and Disadvantages
One major advantage of the DLL protocol is the constant page table size for a given system memory size. As mentioned earlier, page tables of protocols that use sets or trees vary dynamically in size, making these protocols difficult to implement. In DLL, however, page movement is kept track of by using the N-links and P-links, which only require a constant amount of memory in the page tables.

Second, in the DLL protocol, the responsibilities of invalidating a page are distributed by the chain of P-links to all clusters that have copies of the page. This prevents the large burst of messages generated by the owner during invalidations.

Moreover, only one acknowledgment message is sent for each invalidation of a page, instead of one per cluster that has the page as in the DDM algorithm.

Third, in the basic DLL protocol, the cluster that most recently performs a read access to a page becomes the owner of the page. This favors read-modify-write sequences that when a cluster performs write accesses to page immediately after reading from it, it can perform the write-invalidation directly without the need to locate the owner.

Nevertheless, the DLL protocol does have its disadvantages. First, as copies of pages in different clusters have to be invalidated one by one, the time required for the invalidation process is long. This calls for the use of other performance enhancement techniques such as relaxed memory consistency model [5][6] and write-buffering [7], which allow certain memory accesses to be performed before the completion of previous accesses. Second, in the DLL-B protocol, a message travels to the owner by going through a chain of N-links, which grows longer for every read request performed to the page. The time wasted in following the N-link can be substantial as the chain of N-links grows long.
2.4 The DLL Protocol with N-link Reduction

To address the second drawback of the DLL-B protocol discussed in the previous section, the DLL protocol with N-link reduction (DLL-R) was developed to reduce the length of the chain of N-links during read accesses. N-link reduction reduces the length of the chain of N-links during every read request to a page. According to DLL-B, the cluster that generates the read request will become the new owner of the page after the request has been serviced. Therefore, all the clusters that are involved in forwarding the RR message may change their N-links to the requesting cluster, even though the request has not yet been completed. The requesting cluster should lock the page and queue all accesses to it until the RD message is received.

For instance, assume the N-link of cluster c0 points to cluster cl and that of cl points to cluster c2, which is the owner of page p0. Cluster c0 now puts a lock on p0 and sends a RR message to its N-link, i.e., cl, to request read access to p0. When cl receives the RR message, it forwards the message to c2 and at the same time sets its N-link to point to c0, which will become the new owner of p0 after the completion of the read access. Therefore, the N-link of cluster c1 is reduced. Although only the N-links of clusters that are previously involved in forwarding a message are reduced, N-link reduction puts no extra cost to the protocol because it only uses the original RR message without adding new information to it.

3. The DLL Protocol with Stable Owners

The objective of developing the DLL-S protocol is to completely reduce the chains of N-links used in locating the owner of pages. Moreover, it allows multiple read accesses to the same page to be serviced simultaneously by different clusters that have copies of the page.

The memory organization and initialization method of the DLL-S are the same as the DLL-B and DLL-R. The initial order of distribution of the pages is again immaterial to the correctness of the protocol provided that the page table of each cluster is initialized to reflect the initial page placement.

3.1 Memory Access Methods

The read access methods of the DLL-S are different from that of the DLL-B but the write access methods are essentially the same. The read access methods are explained below.

If a page is in an E or S state, the cluster already has a valid copy of the page and thus read accesses to the page can be handled locally. However, as in the DLL-B, when a read access is performed to an I page, the page must be obtained from another cluster that has a valid copy of the page. It sends an RR message to the cluster pointed to by its N-link. If the cluster receiving the RR message does not have a valid copy of the requested page, it forwards the message to its own N-link. Eventually, the RR message reaches a cluster that has a valid copy of the requested page. Note that this cluster may or may not be the owner of the page. When a cluster that has a valid copy of the requested page receives the RR message, it creates an RD message that contains a copy of the requested page and copies of its N-link and P-link and sends it back to the requesting cluster. The requesting cluster changes the state of the requested page to S and its P-link to point to the requesting cluster. When the requesting cluster receives the RD message, it copies the page to its local memory and set the local physical address field of the page table accordingly, changes the state of the page to S and sets its N-link and P-link to the values stored in the RD message. If the received N-link is NULL, meaning the replying cluster is the owner of the page, the requesting cluster sets its N-link to point to the replying cluster. The requesting cluster then completes the read access.

In the process, the requesting cluster is attached to the linked list of clusters joined by their P-links. The N-links of all clusters in this list point to the owner of the page, i.e., the head of the linked list. This arrangement is to facilitate the write request and write invalidate operations, which must still be serviced by the owners.

The following is an example of a read request of the DLL-S. Assume a small system with 3 clusters, c0-c2, with c0 being the initial owner of page p0 and c1 has just completed a read request to p0. The current page table entry of p0 in each cluster is as shown in Table 2.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>State</th>
<th>P-link</th>
<th>N-link</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>S</td>
<td>c1</td>
<td>NULL</td>
</tr>
<tr>
<td>c1</td>
<td>S</td>
<td>NULL</td>
<td>c0</td>
</tr>
<tr>
<td>c2</td>
<td>I</td>
<td>NULL</td>
<td>c0</td>
</tr>
</tbody>
</table>

Table 2. Initial state of p0 in each cluster

Now, another cluster, c2, performs a read access to p0. Since it does not have a copy of p0, it sends an RR message to its N-link, i.e., c0. When c0 receives the RR message, it again replies by sending an RD message that contains a copy of p0 and the value of its N-link (NULL) and P-link (pointing to c1 by the last example) back to c2. Then, it sets its P-link to point to c2. When c2 receives the RD message, it copies the page to its local memory and sets its N-link to point to c0 and P-link to point to c1 according to the value in the RD message. At this point, the read access is completed. The process is depicted in Figure 6. Note that c2 is now part of the linked list in which c0 is the head and c1 is the tail. The state of the page table in these clusters are summarized in Table 3.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>State</th>
<th>P-link</th>
<th>N-link</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>S</td>
<td>c1</td>
<td>NULL</td>
</tr>
<tr>
<td>c1</td>
<td>S</td>
<td>NULL</td>
<td>c0</td>
</tr>
<tr>
<td>c2</td>
<td>I</td>
<td>NULL</td>
<td>c0</td>
</tr>
</tbody>
</table>

Table 3. Final state of p0 in each cluster

3.2 Significance of the DLL-S Protocol

The DLL-S protocol shows two improvements over the DLL-B and DLL-R. First, all clusters in the system that have copies of a certain page can service a remote memory access to that page so the number of read accesses that can be serviced simultaneously is equal to the number of clusters that have copies of the page. This exploits more parallelism.

Second, in the DLL-S, all clusters in a linked list have their N-links pointing to the owner of the page. This is not only an improvement over the DLL-B, in which a message must go through a chain of N-links to reach the owner, but also an
improvement over the DLL-R, in which the chain of N-links is
only partially reduced.

The only additional cost of the DLL-S is that the RD
message now contains the value of the N-link and P-link of the
replying cluster, in addition to the copy of the requested page.
This addition (several bytes), however, is very small compared to
the usual page size (hundreds or thousands of bytes) and can
therefore be justified. Nonetheless, as the ownership of pages is
no longer changed by read accesses, the advantage of the fast
read-modify-write sequence in the basic DLL protocol is lost.

4. Theoretical Analysis of the Protocols

This section compares the DLL-B, DLL-R and DLL-S
protocol, and the DDM algorithm theoretically with respect to the
areas including remote memory access latency, page table
size and message distribution.

4.1 Remote Memory Access Latency

The remote memory access latency \( T_{\text{total}} \) is defined as the
time interval between the issue of a remote memory access and
the completion of that access. It can be divided into 3 parts,
namely the send time \( T_s \), the invalidation time \( T_i \) and the
reply time \( T_r \). The software overhead and queuing delay for
remote memory accesses is approximately the same for the
protocols discussed and therefore will not be included in the
comparison.

4.1.1 Send Time

The send time is defined as the time required for the
memory access request message to travel from the requesting
cluster to the cluster that will service the request. In order to
evaluate the send time, one must first understand the concept of
cycles of accesses.

Define one 'cycle of accesses to a page' to be all the
accesses performed to the page during two invalidation
operations of that page. For instance, cluster \( C_0 \) performs a
write access to page \( p_0 \), thus invalidating all copies of \( p_0 \) in
other clusters; then, all clusters performs read access to \( p_0 \) to
read the value written by \( C_0 \); finally, \( C_0 \) writes to \( p_0 \) again —
this is considered as one cycle of access.

The significant of cycles of accesses in the analysis of remote
memory access latency is that if a certain cluster skip a
cycle of accesses to a certain shared page, a maximum of one
additional step will be necessary for any messages from that
cluster to reach their destination. Consider the following example:
Cluster \( C_0 \) performs a read access to page \( p_0 \) in cycle
0; then, another cluster \( C_1 \) performs a write access to page \( p_0 \),
thus ending cycle 0 and starting cycle 1. Cluster \( C_0 \) does not
perform any accesses to page \( p_0 \) before yet another cluster \( C_2 \)
performs another write access to page \( p_0 \), thus ending cycle 1
and starting cycle 2. In this case, \( C_0 \) skips a cycle (cycle
1) of page \( p_0 \). Now, if \( C_0 \) performs a remote memory access to
page \( p_0 \), it will send a message to \( C_1 \), which is no longer the
owner of page \( p_0 \) because \( C_2 \) has written to page \( p_0 \). Therefore,
one additional step is needed between \( C_1 \) and \( C_2 \) before the
message can reach its destination. This additional number of
messages needed is only a maximum because there are chances
that the message can reach its destination without going
through all the steps. For example, if the second cluster that
writes to page \( p_0 \) is \( C_1 \) instead of \( C_2 \), then \( C_0 \)'s remote memory
access message will reach its destination in only one step.

By the above argument, the following can be deduced. If a
cluster skips \( n \) cycles of accesses to a certain page, a
maximum of \( n \) additional steps will be required for the request
message of a remote memory access performed by that cluster
to reach that page to reach its destination.

The effect of cycles of accesses applies to the DLL
protocols, as well as to the DDM algorithm. In the following
analysis, we shall assume the clusters never miss cycles of
accesses. In cases where cycles of memory accesses are
skipped, the above rules may be used to estimate the additional
overhead required.

In the analysis, the physical distances between any two
clusters are assumed to be the same in order not to bias the
study to any particular network topology. The time required for
a request message to travel one hop, i.e., from a cluster to
another, is assumed to be \( t_0 \), while the time required for a data
message to travel one hop is assumed to be \( t_d \). Note that both \( t_0 \)
and \( t_d \) depends of the speed and latency of the network, which
in turn depends on the traffic condition of the network, and \( t_d \)
also depends on the system page size.

4.1.1.1 Send time for the DLL-B protocol

According to the DLL-B protocol, the send time is not a
constant but rather a variable depending on the number of
clusters that the request message goes through before it reaches
the owner. As a chain of N-links is used to locate the owner of
a page in DLL-B, we can deduce that the send time \( T_s \) of any
particular remote access is:

\[
T_s^{(\text{DLL-B})} = n \cdot t_0
\]  

In equation (1), the cluster that performs the remote access
request is the \( n \)th cluster to do so in the current cycle of
accesses of the page. Recalling from the DLL-B definition,
assuming that the cluster skip no cycle of accesses, a request
message from a cluster will require only one step to reach the
owner of a page if it is the first read access performed to the
page after the most recent invalidation. Then, for every cluster
that performs read access to the page, the chain of N-links will
grow one step longer.

If, on average, the number of clusters that perform memory
access to the page in one cycle of access equals to \( \tilde{r} \), the
average send time \( \bar{T}_s \) for this particular page will be:

\[
\bar{T}_s^{(\text{DLL-B})} = \frac{1 + 2 + \ldots + \tilde{r} \cdot t_0}{\tilde{r} / 2}
\]  

Notice that the average send time for DLL-B is \( O(\tilde{r}) \). This
means that if more clusters share the same page, the
average send time for that page will be higher. This limits the
scalability and the amount of parallelism of the system.

4.1.1.2 Send time for the DLL-R protocol

In the DLL-R protocol, since the N-links of all clusters that
are involved in forwarding an RR message are updated to point
to the requesting cluster, i.e., the new owner, one would expect
its performance to be better than \( O(\tilde{r}) \). In fact, for clusters
that have not missed any cycle, the send time \( T_s \) is:

\[
T_s^{(\text{DLL-R} \text{ first cluster})} = t_r
\]  

for the first cluster that perform remote access to the page
immediately following its invalidation; and is:

\[
T_s^{(\text{DLL-R})} = 2t_r
\]  

for all clusters that perform remote read access to the page
after the first. Therefore, the average send time for DLL-R for
a page shared by an average of \( \tilde{r} \) clusters is:
\[ T_s(DLL - R) = \frac{1 + 2(\bar{n}_r - 1)}{\bar{n}_r} t_r = 2 - \frac{1}{\bar{n}_r}, t_s < 2t_r. \] (5)

Note that the average send time still increases when the number of clusters sharing a page increases; however, there is an upper-bound of 2 \( t_r \), a major improvement over DLL-B.

### 4.1.1.3 Send time for the DLL-S protocol

In the DLL-S protocol, the owner of a page is not changed by read accesses. Therefore, provided that a cluster has not skipped the previous cycle of accesses, it always knows exactly where the owner of a page is. Hence, the send time is:

\[ T_s(DLL - S) = T_i(DLL - S) = t_r. \] (6)

If a cluster skipped the previous cycle of accesses, additional steps will be required as discussed earlier. However, as any cluster that has a certain page may service the read request to that page, the chance that additional steps are required is small indeed.

### 4.1.1.4 Send time for the DDM algorithm

In the DDM algorithm, the owner of a page is not changed by read accesses, so the send time required is the same as in the DLL-S protocol:

\[ T_s(DDM) = T_i(DDM) = t_r. \] (7)

Note that in the DDM algorithm, only the owner of the page may service a read request.

### 4.1.1.5 Comparison

When comparing the send time, the DLL-S protocol and the DDM algorithm are the clear winner. However, the DLL-S protocol has an advantage here because every cluster that has a copy of a page may service the read request to that page; while in the DDM algorithm, only the owner may service any request. The send time of the DLL-R is slightly poorer than the above two but is still acceptable owing to the 2 \( t_r \) upper-bound.

### 4.1.2 Invalidation Time

The invalidation time is defined as the time between the receipt of the write request by the owner of the requested page and the receipt of all the acknowledgment messages by the requesting cluster. The invalidation time for read accesses is always zero.

#### 4.1.2.1 Invalidation time for the DLL protocol family

The invalidation time of all variations of the DLL protocol is the same. For a remote write request, if this is the \( n_r \)th remote request performed to the page during the current cycle of accesses, the number of copies of the page in the system is \( n_r \). Therefore, apart from the owner’s copy of the page, \( n_r - 1 \) WIF messages plus one WIF message are required to invalidate all other copies of the page. The invalidation time and average invalidation time are thus:

\[ T_i(DLL) = n_r t_r, \quad \bar{T_i}(DLL) = \bar{n}_r t_r. \] (8)

#### 4.1.2.2 Invalidation time for the DDM algorithm

During write invalidation in the DDM algorithm, copies of the page are invalidated in parallel. With one invalidation message and one acknowledgment message for each of them. Therefore, the invalidation time for remote write access is constantly equal to:

\[ T_i(DDM) = T_i(DDM) = 2t_r. \] (9)

### 4.1.2.3 Comparison

From the above analysis, the DDM algorithm seems to be the obvious winner. However, the effect of network congestion is not taken into account here. In the DDM algorithm, as copies of the page are invalidated in parallel, a large burst of invalidation messages are sent by the owner simultaneously. According to the characteristics of common networks, the latency of the networks rises sharply when the traffic reaches 60-80% of the capacity of the network [8][9]. As a result, \( t_r \) for the DDM algorithm may increase radically when a large burst of messages is generated, thus increasing the overall invalidation time. Our simulation showed that the actual invalidation time of the DDM algorithm could be longer than the invalidation time of the DLL protocol.

### 4.1.3 Reply Time

The reply time of a remote memory access is defined as the time interval between the generation of the data message by the cluster that services the request and the receipt of it by the requesting cluster. The reply time for write invalidation requests is always zero as no data message is generated.

The reply time for all read and write requests is the same for all the protocols discussed and is equal to \( t_r \). It is because the data message is always sent directly to the requesting cluster.

### 4.2 Page Table Size

The page table size is constant for all the DLL family of protocols. There is one record for each page in the system and four fields in each record, namely the state of page, N-link, P-link and local physical address. In the page table of the DDM algorithm, there is also one record for each page in the system and four fields in each record, namely the state of page, probable owner, copy-set and local physical address. The state of page and the local physical address field of the two DSM algorithms are the same. And the N-link field is equivalent to the probable owner field. Therefore, we need only to compare the size of the P-link field to that of the copy-set. Let the memory required to store a cluster ID be one unit. Then the total size of the P-link field in a cluster is \( n_p \), where \( n_p \) is the number of pages in the system. On the other hand, the copy-set is a dynamic memory structure whose size range from zero to \( N \), where \( N \) is the number of clusters in the system. As a result, the worst case total size of the copy-sets in a cluster is \( n_p \cdot N \). In other words, the worst case size of the copy-sets is \( N \) times that of the P-links. For typical application, the number of clusters sharing the same pages could be several hundreds or thousands, meaning the total size of the copy-sets could be several hundreds or thousands times larger than the P-links. Moreover, enough memory must be saved for the copy-sets; otherwise the system may fail by running out of memory.

#### 4.2.1 Message Distribution

In order to look into the message distribution of the protocols, the pattern by which a cluster generates message is analyzed. In the DLL family of protocols, each cluster usually only generates one message at any one time, except in the case of a WR request, in which the owner generates a WD and a WIF message simultaneously. Therefore, the maximum number of messages generated by a cluster is 2 and the total worst case number of messages generated in the whole system simultaneously is \( 2N \), which is \( O(N) \).

For the DDM algorithm, the maximum number of messages generated by a cluster is \( N + 1 \), which occurs when the owner of a page services a write-request and has to send a data message plus \( N - 2 \) invalidation messages to invalidate
every other cluster's copy of the page. Therefore, the worst case number of messages generated in the whole system simultaneously is \( N \cdot (N - 1) \), which is \( O(N^2) \).

Although it is the worst case situation, several insights can be gained from the above analysis. First, a cluster in the DDM algorithm can generate many more messages than a cluster in the DLL protocols, which means high probability of congestion at that part of the network. Second, in general, the number of messages in the network at any one time would be higher for the DDM algorithm than for the DLL protocols, requiring a higher-bandwidth network. Third, owing to the burst nature of the messages generated by the write invalidation operations in the DDM algorithm, it highly favors a broadcast or multicast network; on the other hand, the DLL protocol works well with any kind of network.

5. Simulated Performance Evaluation

The simulations are implemented as user level programs in a network of workstations running PVM 3 [10]. The network transfer rate of 0.8 byte/cycle (equivalent to 40MB/s on a 50MHz system) and message passing latency of 500 cycles are assumed [11]. The page size is set to be 1kbyte for all three algorithms. In various studies of interconnection network performance, the latency is shown to rise sharply when the network becomes saturated [8][9].

Two common applications, namely the linear equation solver and the matrix multiplier, are used in the simulations to evaluate the performance of the DSM protocols. The linear equation solver solves 256 equations by the Gauss-Seidel method [12] which is an iterative method — the results are repeatedly read, recalculated and written back to the shared memory. Therefore, there are a large amount of read-modify-write sequences involved. The matrix multiplier multiplies two 64 x 64 square matrices by reading the corresponding elements, multiplying and adding the results, then writing the results back to the shared memory. The number of shared memory accesses in the matrix multiplier is much smaller than in the linear equation solver, and there is no read-modify-write sequence. These programs are written with the assumption of a true shared memory, i.e., the distributed nature of the system is hidden. Systems of up to 16 clusters are simulated.

Figure 7 depicts the plot of the speedup for the linear equation solver, in which the speedup is obtained by:

\[
\text{speedup} = \frac{\text{Process time by 1 cluster}}{\text{Processing time by } N \text{ cluster}}.
\]

With a speedup of 4.07 with 16 clusters, the DLL-S protocol is the best performer, although the performance of the DLL-R protocol is very close to that of the DLL-S protocol. With the shorter remote memory access latency and parallel read accesses offered by the DLL-S as discussed before, one would expect a greater improvement over the DLL-R. However, as there are many read-modify-write sequences in the linear equation solver, the DLL-B and DLL-R protocols with their quick read-modify-write property have a greater advantage, and therefore they perform better. In this case, the DDM algorithm achieves only a speedup of 1.55 and 3.35 with 8 and 16 clusters, which are 39.6% and 17.5% less than the DLL-S. This is mainly because of the network congestion and increase in network latency caused by the bursts of invalidation messages.

In order to understand further the impact of the quick read-modify-write feature, the total number of control messages, i.e., messages that do not contain memory page data, and the total number of data messages used are plotted for the DLL-R and the DLL-S protocols in Figure 7 and Figure 9. From the plots, the DLL-R protocol uses 26% more control messages but 15% fewer data messages than the DLL-S protocol. The extra control messages are used by the DLL-R protocol to go through the N-links, which are only partially reduced. However, to explain the larger number of data messages used by the DLL-S protocol, one has to look into the details of the iterative method used in the linear equation solver. In each iteration of the solver, the results from the previous iteration are read from the shared memory, and the new results are written back to the memory after some calculations. In the DLL-R, after the results are read, the cluster becomes the owner of the page in which the results are stored, so when the results are written back to the memory, the WI request can be service immediately and no data message is involved in the write accesses. In the DLL-S, however, reading the results from the memory does not give the ownership of the page to the cluster, so when the results are written back, a WI request is sent to the current owner of the page. If before the WI request is serviced, the cluster receives a WIF message (from another cluster also trying to write the iteration results to the page), thus invalidating its copy of the page, the previous WI request have to be aborted and a new WR request, which involves a data message transfer, is generated. As a result, the number of data messages used by the DLL-S protocol is
messages exist in the network for a longer time. The fact that chiefly because there is no read-in-oId-wrItc sequence in the clusters. The difference between the DLL-S protocol and the parallel read accesses feature. which services more RR the DLL-S protocol generates more messages at one time than chains of N-link to reach the owner. which makes these close to the the DLL-B and DLL-R protocols is not exploited. This is also the matrix multiplier. With a speed of up of 5.82 with 16 clusters, the DLL-S protocol is the most scalable of the four DSM protocols discussed. With an average of more than 8 messages in the system at any one time. the DDM algorithm generates more messages in the system equal to 5.1, which is larger than the other the DLL protocols. It is because remote memory access messages have to go through long chains of N-link to reach the owner. which makes these messages exist in the network for a longer time. The fact that the DLL-S protocol generates more messages at one time than the DLL-R protocol and yet has better performance is due to its parallel read accesses feature, which services more RR messages at one time.

Figure 11 depicts the speedup obtained by the protocols for the matrix multiplier. With a speed up of 5.82 with 16 clusters, the DLL-S protocol is again the best performer, followed by the DLL-R protocol, which has speed up of 5.32 with 16 clusters. The difference between the DLL-S protocol and the DLL-R protocol is larger in this case — the DLL-S protocol is 9.4% faster than the DLL-R protocol in the 16 clusters case — chiefly because there is no read-modify-write sequence in the matrix multiply so the quick read-modify-write advantage of the DLL-B and DLL-R protocols is not exploited. This is also the reason of the performance of the DLL-B protocol being so close to the DDM algorithm.

Finally, from the trend of the graphs, the differences between the performance of the DLL-S protocol and the other three protocols are predicted to be even greater when the number of clusters used is larger than 16. This indicates that the DLL-S protocol is the most scalable of the four DSM protocols discussed.

6. Conclusions

The DLL protocol is a memory efficient, scalable, high-performance and yet easy to implement protocol to maintain memory coherence in DSM systems. In this paper, the DLL protocols with stable owners is introduced and its performance compared, both theoretically and by simulation, to the basic DLL protocol and the DLL protocol with N-link reduction, as well as the DDM algorithm. From the results it appears that the DLL-S protocol has superior performance to the others.

However, one drawback of the DLL-S protocol as compared to the DLL-B and DLL-R protocol is that it does not have the advantage of quick read-modify-write, which turns out to affect its performance to some extent. This fact also suggests that different applications may favor different DSM protocol and therefore, a protocol can never be absolutely the best.

References


