Dynamic Snooping in a Fault-Tolerant Distributed Shared Memory

Larry Brown and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Abstract

Distributed shared memory (DSM) allows multicomputer systems with no physically shared memory to be programmed using a shared memory paradigm. However, as the number of nodes in a system increases the probability of a failure that can corrupt the DSM increases. This paper presents a fault-tolerant DSM (FTDSM) algorithm that can tolerate single node failures. Each page in the DSM is assigned a snooper that keeps a backup copy of the page and can take over if the page owner fails. The snooper is dynamic because the responsibility for snooping a page can migrate from node to node. The FTDSM presented in this paper is an improvement over other FTDSMs because it is scalable, is based on the efficient dynamic distributed manager (DDM) DSM algorithm, does not require the repair of a failed processor to access the DSM, and does not query all nodes to rebuild the state of the DSM. It is shown that any single node failure can be tolerated because either the owner or the snooper of a page can always be found.

1 Introduction

Parallel and distributed computing systems are becoming increasingly important. These systems can achieve better response time and can have higher system availability than uniprocessors. Multicomputer or message-passing systems without physically shared memory can be scaled to large numbers of nodes because there is no contention for shared memory and scalable interconnection networks can be used to connect the nodes. However, programming these systems is difficult because the interaction and movement of data between processes must be explicitly controlled. It is important to develop a paradigm for communications that is both efficient to implement and easy to program. One important paradigm is distributed shared memory (DSM) [3, 5, 6]. DSM presents the programmer of a scalable distributed-memory system with a familiar shared-memory interface.

Various DSM algorithms have been developed. Many of these algorithms use ownership-based write-invalidate protocols to enforce consistency. However, most DSM algorithms are not fault-tolerant. Because the state of a DSM system is distributed among all nodes in the system, the failure of a single node can cause the entire system to fail. If a node fails access to a group of pages may be lost or information may become inconsistent. As the number of nodes participating in the DSM grows, the probability of a failure increases, making fault-tolerance more desirable.

Other reliable, recoverable, or fault-tolerant DSM (FTDSM) systems have been proposed. [10] describes a FTDSM that uses a central disk server on a LAN to store process checkpoints and the state of the DSM. To ensure consistency, all dirty pages are checkpointed whenever a dirty page is migrated to another node. This ensures that all changes made by a node become visible as soon as any changes become visible. If the node then fails, changes that causally precede changes to the migrated page will not be lost. The system is based on a centralized manager DSM algorithm. In [9] the state of the DSM is treated as a distributed database. Each step in the DSM algorithm is treated as a transaction which is decomposed into subtransactions that can be committed unilaterally. To ensure consistency, message logs are flushed to stable storage whenever a dirty page is migrated to another node. The system is based on a fixed distributed manager DSM algorithm. In [8] a system is described in which a node keeps a copy of page when it migrates ownership of the page to another node. If the migrated page is dirty, copies of all other dirty pages owned by the node are also sent to the new owner. This ensures that all changes made by the old owner become visible as soon as any changes become visible. If the new owner fails, the node with the most recent copy of the page can be located. The system uses broadcasting to locate page owners and control the DSM algorithm.

1063-6927/94 $03.00 © 1994 IEEE

In these FTDSM systems recovery after a failure may require querying all other nodes, or DSM pages may be unavailable until a processor recovers. The FTDSM developed in this paper differs in that it avoids the need to consult every node to recover after a failure and does not block until a processor recovers. The system presented here is based on the dynamic distributed manager (DDM) DSM algorithm [5] which is more efficient and scalable than centralized manager, fixed distributed manager, or broadcast-based DSM algorithms.

The system developed in this paper is called the integrated-snooper (IS) FTDSM. Each page has a snooper node that monitors the activities of the page owner. The responsibility for snooping a page can migrate from node to node; the snooper of a page is dynamic. The snooper keeps track of the page contents, location of page replicas, and the identity of the page owner. The snooper can respond on behalf of a failed owner. The term snooper is used because the IS FTDSM is derived from other snooper-based systems [2] in which separate nodes are dedicated to snooping on a broadcast network. In the IS FTDSM a node can both snoop some pages and own other pages. Thus the ability to own and snoop pages is integrated at every node. The IS FTDSM is not restricted to a broadcast network; scalable interconnection topologies such as hypercubes and meshes can be supported.

This paper presents the IS FTDSM algorithm, discusses how the snooper of a page is located, and shows that the IS FTDSM tolerates single node failures. Each node maintains knowledge of the last known owner and snooper of each page. Properties of this knowledge are studied and used to prove that the information stored by a failed node can always be recovered. Section 2 describes the DDM DSM algorithm upon which the IS FTDSM is based. Section 3 presents the IS FTDSM, and recovery after a node failure is discussed in Section 4. Section 5 summarizes the paper and suggests future work. Due to space limitations, not all proofs are given in this paper. The proofs can be founds in [2].

2 Dynamic Distributed Manager DSM Algorithm

One of the most widely applicable and better performing DSM algorithms is the DDM algorithm of IVY [5]. A DDM DSM system consists of a set of identical nodes. Each node contains a processor and local memory. The nodes are joined by an interconnection network. Regardless of the physical topology of the interconnection network, the logical topology is fully connected. Routing is performed by the underlying communication system, and the details of its implementation are not considered here. A logical channel exists between every pair of nodes. These channels are assumed to be FIFO and error-free. Messages from the same node arrive in the order sent, and no messages are lost, duplicated, or corrupted.

The shared memory space is divided into a set of fixed size pages. Before the system starts each page is assigned to be owned by one of the nodes. Initially there is one copy of each page, and that copy is at the owner. All nodes can determine the initial owner of each page. The number of copies and ownership of pages changes as the system runs. Each node maintains a page table indicating the status of each page in the DSM. The page table consists of the following fields for each page: probowner – the probable owner of the page, copyset – a list of nodes that have a read-only copy of the page (kept only by the page owner), and present – true if a copy of the page is in local memory.

The probowner field indicates the last known owner of a page. That owner may have since given ownership to another node. The probable owner will either still own the page or will be able to forward the request in the direction of the owner. The probowner fields form a chain from node to node that eventually leads to the owner. Whenever a message is received from which the owner of a page can be deduced, the probowner field is updated.

The copyset field is used by the page owner to keep track of the location of replicas of a page. This information is used to invalidate the replicas before writing to the page. Whenever the owner distributes a copy of a page, it adds the identity of the node requesting the page to the copyset. Figure 1 shows a snapshot of the state of a DDM DSM and the corresponding page tables for each node. In the figure a probable owner chain for page p_3 leads from node n_1 to n_3 to n_2. p_3 is owned by n_2, and the copyset for p_3 indicates that there is one copy of p_3 at n_3.

When a node reads a page that is not cached in local memory, the memory management hardware generates a read fault. The read fault handler recognizes the page as a DSM page. A read request is sent along the probable owner chain. When the request reaches the owner, the owner returns a read-only copy of the page and records the identity of the requesting node in the copyset. When the requester receives the page, it updates its probowner field to be the node that re-

Page	Pres.	Probowner	Copyset	Page	Pres.	Probowner	Copyset	Page	Pres.	Probowner	Copyset
1	1	1	empty	1	0	1	-----	1	0	1	-----
2	1	1	2,3	2	1	1	-----	2	1	1	-----
3	0	3	-----	3	1	2	3	3	1	2	-----
4	0	2	-----	4	1	2	empty	4	0	2	-----

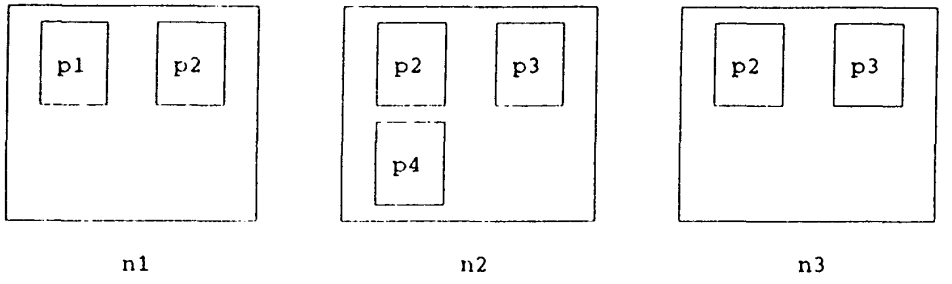

Figure 1: Snapshot of a DDM DSM.

turned the page.

Before a node writes to a page it must first become the page owner and invalidate all copies of the page. A write fault is generated if the page is not present in local memory or is present in read-only mode. If it is not already the page owner, the node sends a write request to the probable owner. Each node that forwards the write request updates its probowner field to be the requester because the requester will become the new owner. Any requests for the page which arrive at the new owner before it actually receives ownership are queued and serviced after the transfer of ownership is complete. The owner returns a copy of the page and the copyset of the page. The requester is now the new page owner. The previous owner invalidates its copy of the page and updates its probowner field to be the new owner. If the copyset is not empty the owner has read-only access to the page. To enforce sequential consistency [4], the owner must ensure that it has the only copy of the page before writing to it. The owner sends an invalidate message to each node in the copyset. When a node receives an invalidate message it invalidates the page, updates the probowner field, and returns an acknowledgement to the owner. When all invalidate messages have been acknowledged the owner can write to the page.

The DDM algorithm consists of two parts, a page fault handler which handles faults for DSM pages and a message handler which handles incoming requests sent by other nodes. In the DDM algorithm presented below probowner(p), copyset(p), and present(p) refer to the probable owner, copyset, and present bit of page p.

```
page_fault_handler (fault f, page p)
{
switch (f) {
  case read_fault:
    send read to probowner(p);
    receive read_reply;
    probowner(p) = originator of read_reply;
    break;
  case write_fault:
    if (probowner(p) != ME) {
       send write to probowner(p);
       receive write_reply;
       probowner = ME;
    }
    for (node in copyset(p)) {
        send invalidate to node;
        receive invalidate_reply from node;
    }
    copyset(p) = empty;
    break;
}
}

message_handler(message m)
{
switch (message_type) {
  case read:
    if (probowner(p) != ME)
       forward read to probowner(p);
    else {
       copyset(p) = copyset(p) + requester;
       send read_reply to requester;
    }
```

```
    break;
  case write:
    if (probowner(p) != ME) {
       probowner(p)=requester;
       forward write to probowner(p);
    } else {
       probowner(p) = requester;
       present(p) = 0;
       send write_reply to requester;
    }
    break;
  case invalidate:
    present(p) = 0;
    probowner(p) = originator;
    send invalidate_reply;
    break;
}
}
```

The DDM algorithm requires all nodes to be operational. The failure of any node can corrupt the DSM, causing the entire system to fail. A node failure can result in loss of the only copy of a page, breaks in the probable owner chain, and loss of replica location information.

3 Integrated-Snooper FTDSM

To implement a highly-available FTDSM based on the DDM algorithm three pieces of information must always be available for each page: the identity of the page owner, the copyset of the page, and the contents of the page. If the owner fails ownership must be migrated to a new owner in order for the page to remain available to other nodes. If the owner fails when no other node has a copy of the page, the contents of the page are lost. To invalidate a page all nodes with a copy of the page must be found, but if the owner fails the copyset is lost. The owner is found by following a chain of probable owners. If a node in the chain fails the chain is broken, and the nodes below the break cannot locate the owner. It is desirable that these failures be repaired without querying every node and without waiting for a failed node to recover. The IS FTDSM must maintain the coherency of each page (read returns the latest value written) and sequential consistency between pages (causally preceding modifications are not lost).

It is assumed that the nodes of an IS FTDSM system exhibit fail-stop failure semantics [7]. The memory of a node is volatile and does not survive failures. When a node fails the contents of its memory, including DSM pages and the DSM page table, are destroyed and are inaccessible to other nodes.

In the IS FTDSM each page is assigned a snooper node which monitors the activities of the owner and can respond for the owner if the owner fails. The snooper monitors the page contents, location of page replicas, and the identity of the page owner as ownership migrates from node to node. The IS FTDSM is derived from FTDSM systems [2] in which one or more nodes dedicated to snooping snoop traffic on a broadcast network. The nodes that do not snoop, but perform application processing, are called workers.

In the IS FTDSM nodes are not dedicated to snooping, but also perform application processing. Every node can be both a worker and a snooper. Snooping and application processing is integrated at every node. A node may be referred to as a worker or a snooper to emphasize its role in a particular context. To allow a more flexible and scalable configuration, the IS FTDSM does not require a broadcast network. On a nonbroadcast network a node cannot physically snoop every message sent by every node. Messages that must be snooped are sent twice, once to a worker and once to a snooper. The message sent to the snooper need not always contain all of the data in the original message.

Initially, each node is assigned ownership of a set of pages so that each page is owned by one and only one node. Each node is also assigned responsibility for snooping a set of pages so that each page is snooped by one and only one node and no node owns and snoops the same page. Every node knows the owner and snooper of every page when the system is initialized.

The owner and snooper of a page must never be the same node. Otherwise, the failure of that node could cause the only copy of the page to be lost. Before becoming the owner of a page it is snooping, a node must migrate responsibility for snooping the page to another node.

Page ownership is dynamic and pages are located by following the probable owner chain. Like the page owner, the page snooper is also dynamic. The snooper of a page changes if the node that snoops a page becomes the page owner. A probable snooper field, probsnooper, is added to the DSM page table. When the DSM is initialized the probsnooper field points to the actual snooper of the page. As the computation progresses and responsibility for snooping migrates, the probsnooper field might not point to the actual snooper of a page, rather it points to the last known snooper of the page. The probsnooper fields form a chain leading to the snooper just as the probowner

fields form a chain leading to the owner.

Before the page owner modifies a page, no other worker has a copy of the page because all copies have been invalidated. The snooper keeps a copy of every page, but modifications made by the page owner are not made visible to the snooper until the owner sends the page to another worker. The owner's modified copy of the page is said to be dirty, and no other node, including the snooper, is aware of the modification. A bit can be added to each page table entry to indicate if a page is dirty. If the owner fails after modifying a page, but before sending the dirty page to another node, the modification has no effect. ¿From the point of view of the other nodes it appears that the owner failed before making any modifications.

A worker may modify several pages that it owns before any of the changes become visible to other nodes. Before the worker migrates (or replicates) a dirty page that page, and all other dirty pages it owns, must first be backed up by the snoopers of those pages. This ensures that all changes made by a worker become visible to the snoopers when any of the changes become visible to another worker, even if the worker fails immediately after migrating a page. Otherwise, a page modified before the migrated page was modified, and which causally affects the migrated page, could be lost if the worker fails, violating sequential consistency.

All dirty pages must be backed up as an atomic unit; either all of the pages are backed up, or none of them are backed up. Backing up all dirty pages owned by a worker whenever any dirty page is migrated is called page flushing. The technique of atomically flushing all dirty pages is analogous to techniques used in cache-based recovery schemes such as CARER [1] in which a checkpoint is established and all dirty cache lines are flushed to memory whenever any dirty cache line must be written to memory. The FTDSM systems [8, 9, 10] described above use similar techniques.

When multiple dirty pages must be flushed more than one snooper may need to receive pages. Atomic agreement among the participating snoopers is necessary to protect against failure of the flushing worker. A two-phase commit (2PC) protocol can be used to ensure that the page flush is atomic. Each dirty page must be sent to the responsible snooper, and the participating snoopers must agree that all the dirty pages have been received from the worker. Several variations of 2PC page flushing algorithms in which the worker or one of the snoopers coordinate the 2PC protocol are discussed in [2].

Figure 2 shows a snapshot of an IS FTDSM. The probsnooper field has been added to the page table. When a node is the snooper of a page it knows the identity of the page owner and keeps a copy of the page contents and the copyset. In Figure 2 node n_2 snoops page p_1. The copyset of p_1 is empty, and the owner has modified the page. The page is dirty but has not been flushed to the snooper. The snooper can also be in the copyset of the page it snoops; n_3 is in the copyset of p_2. Even though the snooper has a copy of the page, it would be incorrect for the worker on that node to access the page without executing the normal DSM algorithm. If the snooper supplies the page to the worker without the owner's knowledge, the worker will not be in the copyset. Also, the owner might have written to the page, in which case the copy at the snooper would be out of date.

The owner always knows the identity of the snooper. This is necessary to coordinate page flushing, to make snooping more efficient, and for recovery after a failure. In order to determine which snoopers are involved in a page flush the owner must know the snooper of each page to be flushed. The owner must send messages to the snooper so the snooper can monitor the owner. This is more efficient when the owner knows the identity of the snooper because these messages can be sent directly to the snooper without following the probable snooper chain.

To ensure that a new owner knows the identity of the snooper, write reply messages are augmented to contain the identity of the snooper. Also, when the snooper migrates responsibility for snooping a page to another node the owner is informed of the identity of the new snooper. Read reply and invalidate messages are also augmented to contain the identity of the snooper. This is required for correctness of the recovery algorithms described in Section 4.

The integrated snooper FTDSM algorithm is divided into two threads of control. One thread implements the snooper; the other thread implements the worker. The algorithm for the snooper thread is presented below:

```
snooper_message_handler(message m)
{
switch (message_type) {
  case read:
  case write: // probowner chain is broken
    if (probsnooper(p) != ME)
       forward request to probsnooper(p);
    else {  // I am the snooper
       forward request to probowner(p);
       if (failed(probowner(p)) {
          probowner(p) = requester;
```

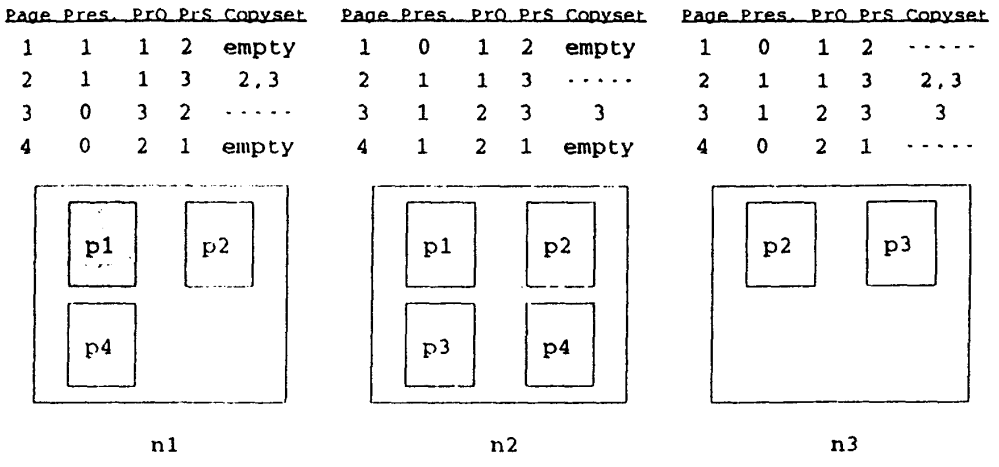

Figure 2: Snapshot of an IS FTDSM. PrO–probowner, PrS–probsnooper.

```
            send write_reply to requester;
          }
        }
        break;
      case flush: // worker is flushing pages
        engage in 2PC with other snoopers;
        break;
      case add_to_copyset: // sent by owner
        //before sending read reply to requester
        copyset(p) = copyset(p) + requester;
        break;
      case remove_from_copyset: // sent by owner
        // after receiving an invalidate reply
        copyset(p) = copyset(p) - requester;
        break;
      case new_owner: // sent by owner before
        // sending write reply to requester
        probowner(p) = requester;
        break;
      case snoop_page: // used to migrate
        // responsibility for snooping
        probsnooper(p) = ME;
        break;
  }
  }
```

The worker thread implements the DDM algorithm except that before migrating a dirty page all dirty pages must be flushed; if a request is forwarded to a failed node the request is resent to the probable snooper; if the worker needs to write a page it is snooping it must migrate responsibility for snooping the page; before sending a read reply to another node a message indicating that a node is being added to the copyset must be sent to the snooper; after receiving an invalidate reply from another node a message indicating that a node is being removed from the copyset must be sent to the snooper; and before sending a write reply to another node a message containing the identity of the new owner must be sent to the snooper. A more detailed description of the algorithm can be found in [2].

4 Recovery

The failure of a node results in the failure of both the worker and the snooper threads of control at that node. When a node fails, other nodes must take over its role, both for the pages it snooped and for the pages it owned. This ensures that there are always two copies of every page stored at failure independent nodes.

Node failures are detected by timeouts in the communication system. When a node detects the failure of another node, it broadcasts a failure announcement. There are four types of information that must be recovered when a node fails: new owners must be found for pages owned by the failed node, new snoopers must be found for pages snooped by the failed node, probable owner chains passing through the failed node must be repaired, and probable snooper chains passing through the failed node must be repaired.

A node can detect that it is responsible for snooping a page owned by the failed node because the

snooper always knows the identity of the owner. The snooper must migrate ownership of the page to another node. Similarly, a node can detect that it owns a page snooped by the failed node because the owner always knows the identity of the snooper. The owner can migrate responsibility for snooping the page to another node.

When a node fails it can also affect pages that it does not own or snoop. All probable owner and probable snooper chains passing through the failed node are broken. When a probable owner chain is broken it can be repaired if the probable snooper chain can be followed to the snooper. The snooper can return the identity of the owner, and the broken link can be replaced by a link pointing to the owner. When a probable snooper chain is broken it can be repaired if the probable owner chain can be followed to the owner. This requires that the probable owner and probable snooper chains are edge disjoint. The owner can return the identity of the snooper, and the broken link can be replaced by a link pointing to the snooper.

The broken chain must be repaired before a second node failure occurs, perhaps breaking the other chain. Otherwise, it would be necessary to query all nodes to find the owner or snooper. When a node learns of the failure of another node, it checks the probowner and probsnooper fields of each page in the page table. Any fields that point to the failed node must be recovered by sending a message along the unbroken chain. For example, if the probable owner chain is broken a message asking for the identity of the owner is sent along the probable snooper chain to the snooper.

Each chain depends on the other for its recovery. That is, no single failure should be able to isolate a node from both the snooper and the owner of a page. This ensures that the healthy chain can be used to repair the broken chain. To show that this is always possible given any single node failure in the IS FTDSM, several important properties of the probable owner and probable snooper chains are given. Then the concept of 1-failure independence of the chains is defined, and the chain properties are used to show that the chains are 1-failure independent.

The probable owner fields of all nodes for a given page form a directed graph G_o. Except for the edge from the owner to itself, G_o can be thought of as a directed tree with the owner at the root. Let $P_o^k(n_i)$ be the node pointed to by the probowner field of page k on node n_i. For page p_k, $G_o = (N, E_o)$ where N, the set of nodes, are the vertices, and $E_o = \{e_{ij} \mid P_o^k(n_i) = n_j\}$ is the set of edges.

The probable snooper fields for a page also form a directed graph, G_s, that is similar to G_o. Let $P_s^k(n_i)$ be the node pointed to by the probsnooper field of page k on node n_i. Note that the subscript s stands for the probable snooper and the subscript o stands for the probable owner. For page p_k, $G_s = (N, E_s)$ where N, the set of nodes, are the vertices, and $E_s = \{e_{ij} \mid P_s^k(n_i) = n_j\}$ is the set of edges.

G_o and G_s represent the probable owner and probable snooper chains for one page of the DSM. There is a separate graph for each page. This discussion applies to all pages concurrently, but for clarity, only one page is considered.

Let $T(P_o^k(n_i))$ represent the latest time at which the knowledge n_i has of the owner of page p_k is known to be correct. Let $T(P_s^k(n_i))$ represent the latest time at which the knowledge n_i has of the snooper of page p_k is known to be correct. Since it is clear that relations such as $>$ apply to the time at which some knowledge is known to be true, and not to the probable owners or probable snoopers themselves, statements such as $T(P_o^k(n_i)) > T(P_s^k(n_i))$ are usually written as $P_o^k(n_i) > P_s^k(n_i)$.

The following lemmas state relationships between the times the knowledge nodes have of the owner and snooper of a page are known to be correct. These lemmas are used to prove that the probable owner and probable snooper chains are failure independent. The first lemma states that each nonowner one step further along a probable owner chain has more recent knowledge of the owner than the preceding worker on the chain.

Lemma 1 $\forall i, k(P_o^k(P_o^k(n_i)) \neq (P_o^k(n_i)) \rightarrow P_o^k(P_o^k(n_i)) > P_o^k(n_i))$

Proof: When n_i discovers the identity of the owner, $P_o^k(n_i)$ points to the root of G_o. At this time, say time t, $P_o^k(n_i)$ is the owner. $P_o^k(n_i)$ could then migrate ownership to a node other than n_i, $P_o^k(P_o^k(n_i)) \neq n_i$. This migration must take place after time t since the owner cannot inform one node that it is the owner and migrate ownership to another node in one step of the algorithm. Also, n_i is not aware of the migration. Therefore, the knowledge of the owner at $P_o^k(n_i)$ is more recent than the knowledge of the owner at n_i; $P_o^k(P_o^k(n_i)) > P_o^k(n_i)$. The restriction $P_o^k(P_o^k(n_i)) \neq (P_o^k(n_i))$ limits n_i to level 3 or greater (root is at level 1). Otherwise, there are not enough ancestors of n_i for the lemma to apply. □

An analogous lemma can be stated for the probable snooper chain.

Lemma 2 $\forall i, k(P_s^k(P_s^k(n_i)) \neq (P_s^k(n_i)) \rightarrow P_s^k(P_s^k(n_i)) > P_s^k(n_i))$

The following two lemmas show that some time relationships hold between G_o and G_s. The first lemma states that if a node was the owner, its knowledge of the snooper must be at least as recent as the time another node knew it was the owner. The second lemma states that if a node was the snooper, its knowledge of the owner must be at least as recent as the time another node knew it was the snooper. The proof of Lemma 4 is analogous to the proof of Lemma 3.

Lemma 3 $\forall i, k(P_s^k(P_o^k(n_i)) \geq P_o^k(n_i))$

Proof: n_i knew that $P_o^k(n_i)$ was the owner at at some time t. Since the owner always knows the identity of the snooper, and $P_o^k(n_i)$ was the owner at time t (or later), $P_o^k(n_i)$ must have knowledge of the snooper at least as recent at time t; $P_s^k(P_o^k(n_i)) \geq P_o^k(n_i)$. □

Lemma 4 $\forall i, k(P_o^k(P_s^k(n_i)) \geq P_s^k(n_i))$

Another important property is that on any node the knowledge of the identity of the owner of a page is always at least as recent as the knowledge of the identity of the snooper of the page.

Lemma 5 $\forall i, k(P_o^k(n_i) \geq P_s^k(n_i))$.

Proof: (outline) The proof is by induction on the number of steps in the IS FTDSM algorithm that change the probowner or probsnooper field of a page on any node. It is shown that the lemma is true after each such step in the IS FTDSM algorithm. □

G_o and G_s have the same vertex set, but their edge sets are disjoint. This property is necessary in order for the probable owner and probable snooper chains to be 1-failure independent. When E_o and E_s are disjoint no node can have equal probowner and probsnooper fields for a page. That is, $\forall i, k(P_o^k(n_i) \neq P_s^k(n_i))$. Otherwise, the failure of $P_o^k(n_i)$ would isolate n_i and all of its descendants from both the owner and the snooper. This property is stated in the following lemma.

Lemma 6 *For a given page in the IS FTDSM* $E_o \cap E_s = \emptyset$.

Proof: (outline) The proof is by induction on the number of steps in the IS FTDSM algorithm that change the probowner or probsnooper field of the page on any node (and thus change E_o or E_s). It is shown that the lemma is true after each such step in the IS FTDSM algorithm. □

The probable owner chain and probable snooper chain must be 1-failure independent. When G_o and G_s are 1-failure independent a probable owner edge that leads to the failed node can be circumvented by following the probable snooper link and vice versa. Define a graph $G_u = G_o \cup G_s = (N, E_o \cup E_s) = (N, E_u)$ which is the union of the probable owner and probable snooper graphs of a page. Let each edge in E_u be augmented with a marker indicating which set, E_o or E_s, it was originally a member of. That is, a node can distinguish between probowner and probsnooper fields. Let G_f^i be the 1-failure graph that is derived from G_u as a result of the failure of node n_i, $G_f^i = G_u - n_i$. G_f^i represents the combined probable owner and probable snooper graphs after the failure of one node. Let $R(G)$ be the transitive closure, or reachability relation, of the relation represented by the digraph G.

Definition 1 *For a given page* G_o *and* G_s *are 1-failure independent if* n_o *is the page owner,* n_s *is the page snooper, and* $\forall i, j (i \neq j \rightarrow n_j = n_o \vee n_j = n_s \vee (n_j, n_o) \in R(G_f^i) \vee (n_j, n_s) \in R(G_f^i)$.

Definition 1 states that G_o and G_s are 1-failure independent if for any single node failure all healthy nodes can send a message to either the snooper or owner of the page by following some combination of probable owner and probable snooper edges. Once the owner or snooper is reached the other can be reached unless it is the node that failed. If the owner or snooper did fail, it will be recreated at a healthy node by the one that did not fail.

The IS FTDSM uses a simple deterministic chain following algorithm which will not always find the shortest path to the owner or snooper, but is easy to implement using only local knowledge. A message follows probable owner (snooper) edges until it reaches the owner (snooper) or encounters a failed node. If a failed node is encountered the message begins following probable snooper (owner) edges. The same type of edge is followed until the owner or snooper is reached or until a failed node is encountered.

The following theorem states that the probable owner and probable snooper chains are 1-failure independent when using the IS FTDSM chain following algorithm.

Theorem 1 *For any page* G_o *and* G_s *are 1-failure independent when using the IS FTDSM chain following algorithm.*

Proof: (outline) Lemma 6 is used to show that a single failure cannot divide G_f^i into two disconnected components. The other lemmas given above are used to show that there are no cycles in G_f^i so progress toward the owner or snooper is made whenever an edge is followed. □

5 Conclusions

This paper has presented the IS FTDSM, a reliable DSM system that is based on the idea of a snooper assigned to each page. The snooper monitors the activity of the page owner and can respond on behalf of a failed owner. The advantages of the IS FTDSM over other FTDSMs are that it is scalable, is based on the efficient DDM algorithm, does not require the repair of a failed processor to access the DSM, and does not query all nodes to rebuild the state of the DSM.

The IS FTDSM can be extended to other ownership-based protocols, such as the type-specific protocols of Munin [3], by assigning a snooper to monitor and backup the owner. The snooper of a page, like the owner, can migrate from node to node. The responsibility for snooping a page must migrate if the snooper becomes the page owner. Otherwise, there would be a single point of failure. The IS FTDSM uses a probable snooper chain to locate the snooper of a page. The properties of the probable owner and probable snooper chains were studied, and it was shown that the chains are 1-failure independent. That is, given any single node failure all nodes can still reach either the owner or the snooper of a page.

Further study is needed to determine the performance and overhead of the IS FTDSM. Some preliminary analysis is presented in [2]. Since each node is both a worker and a snooper, each snooper has, on average, the load of snooping the messages generated by the worker thread of one node. The worker thread generates extra messages (compared to the DDM algorithm) when sending read reply, write reply, and invalidate messages and when flushing pages and migrating responsibility for snooping. Most of these messages are short and cause minimal processing at the snooper. The bulk of the extra work may be caused by page flushing. The frequency of page flushes, number of pages flushed, and number of snoopers involved in page flushes will have a important impact on overall system performance. These characteristics will vary with the page reference behavior of the particular application. An implementation or detailed simulation would allow the performance of a variety of applications to be measured.

Other issues related to the IS FTDSM such as policies for choosing the new snooper when migrating responsibility for snooping and integrating application-level recovery with the IS FTDSM algorithm are discussed in [2].

References

[1] R. E. Ahmed, R. C. Frazier, and P. N. Marinos. Cache-aided rollback error recovery (CARER) algorithms for shared-memory multiprocessor systems. In *Proc. 20th Int. Symp. on Fault-Tolerant Computing*, pages 82–88, June 1990.

[2] L. Brown. *Fault-Tolerant Distributed Shared Memories.* Ph.D. dissertation, Florida Atlantic University, Department of Computer Science and Engineering, Dec. 1993.

[3] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and performance of Munin. In *Proc. 13th ACM Symp. on Op. Sys. Principles*, pages 152–164, Oct. 1991.

[4] L. Lamport. How to make a multiprocessor that correctly executes multiprocess programs. *IEEE Trans. on Comput.*, C-28(9):690–691, Sept. 1979.

[5] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. on Comput. Syst.*, 7(4):321–359, Nov. 1989.

[6] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52–60, Aug. 1991.

[7] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Trans. on Comput. Syst.*, 1(3):222–238, Aug. 1983.

[8] M. Stumm and S. Zhou. Fault tolerant distributed shared memory algorithms. In *Proc. 2nd IEEE Symp. on Parallel and Distributed Processing*, pages 719–724, Dec. 1990.

[9] V.-O. Tam and M. Hsu. Fast recovery in distributed shared virtual memory systems. In *Proc. 10th Int. Conf. on Distributed Computing Systems*, pages 38–45, May 1990.

[10] K.-L. Wu and W. K. Fuchs. Recoverable distributed shared virtual memory. *IEEE Trans. on Comput.*, 39(4):460–469, Apr. 1990.