OPDSM: A Combinatorial Object-Based and Page-Based DSM Model

Milind R. Penurkar
Assistant Professor
Department of Information Technology
M.I.T. College of Engineering, Pune,
Maharashtra, India

Rekha S. Sugandhi
Assistant Professor
Department of Computer Engineering
M.I.T. College of Engineering, Pune,
Maharashtra, India

ABSTRACT
The Distributed Shared Memory (DSM) system is designed on the basis of page-based, shared-variable-based or object-based access. There are certain advantages and disadvantages of each access methodology. Comparison of such different access types in a distributed environment is based on parameters namely, type of operation (data read or update), maintenance of consistent data copies on all nodes accessing shared data, administration of data manipulations on multi-computers, etc. This paper compares the design and implementation issues for page-based and object-based DSM in a multicomputing environment. The paper also proposes a hybrid model for implementation of a DSM system that combines some features of page-based and object-based access that is focused on efficient data access with optimum performance with respect to memory access and cohesive data/object organization. Thus the hybrid model is proposed to get a common function of data sharing by using the best features of both page-based and object-based DSM models.

Categories and Subject Descriptors:
[C.2.4] Distributed Systems: Distributed Applications

General Terms
Design, Theory

Keywords
Distributed Shared Memory, DSM, page-based DSM, object-based DSM

1. INTRODUCTION
Distributed systems have evolved consistently since the past three decades and still has a great scope of optimization with respect to various aspects namely, parallel processing, faster data access and propagation, and effective system control in a de-centralized environment. Distributed systems may be implemented on multiprocessors (with or without caching) as well as on multicomputers (that refer to a network of computers, that each come with their own processor(s) and memory).

Optimum implementation of shared memory systems in a network of multicomputers is more challenging as compared to that on multiprocessors, due to varied capabilities of the node processors,

organization of underlying systems and network issues. Data sharing should be necessarily facilitated in DSM systems in order to run processes in parallel for higher throughput. This can be achieved in one of three ways: page-based, shared-variable-based or object-based. This paper is organized as follows: In section II, brief comparison between the two models namely, page-based and object-based DSM is presented. In section III, the introduction to proposed hybrid model is presented. Section IV discusses the implementations of the proposed model. In section V we discuss the different issues related to the hybrid model. In section VI we give insight to the pros, cons and applications of our model and finally section VII concludes the paper.

2. COMPARISON OF PAGE-BASED DSM AND OBJECT-BASED DSM

2.1 Page-based DSM
In a page-based DSM, the data to be shared among processes are organized as logical fixed-size pages that are distributed over multicomputers. Whenever a page required by a process is locally available, the DSM grants access to the required page via the Memory Management Unit (MMU). This requires simple memory access to the secondary storage that does not involve network access. On the other hand, when a process tries to access a page, that is non-local to the machine, then the page faults to the DSM. The DSM software should now search for the page on a network of computers in a true distributed system and when found is fetched from the source computer. The faulting instruction is then restarted and can now complete [1].

Page-based or block-based DSMs are an extension of traditional Virtual Memory systems thus are usually implemented at the hardware and/or OS layers. Because the implementation is at the H/W and/or OS layers there is complete transparency with respect to 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.

Implementation of page-based DSM at the H/W and/or OS layers needs to modify the existing systems (H/W and/or OS). Such implementations are usually system dependent and hence should have homogenous systems. Existing multiprocessor programs can easily be adopted to run on the page-based DSM. Number of memory consistency models is in existence for the maintaining the consistency among the processor nodes. Generally page invalidation method is used to keep pages consistent among the processor nodes of the distributed system.

False sharing is one of the prominent problems in the page-based memory.

Synchronization is achieved using synchronization managers implemented on different nodes each taking care of locking and

unlocking. Some examples of page-based DSM are IVY, Mermaid and, Blizzard [6].

2.2 Object-based DSM

In object-based DSM, processes on multiple machines share an abstract space filled with shared objects. An object is a programmer-defined data structure consisting of internal data (object state) and procedures (methods/operations) that operate on the object state. Since direct access to the internal object state is not allowed, the property of information hiding forces all references to an object's data to go through the methods that also helps structure the program in a modular way [1]. In object-based DSM, the location and the management of the object is handled automatically by the runtime system.

Implementation of Object-based DSMs is mostly done using the language/compiler or library layers. In Object-based DSM(ODSM) modifications must be

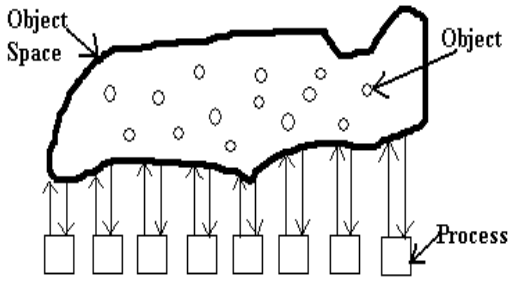

Figure 1 Object-Based DSM

done at user level for its implementation i.e. changes must be done to programming model used and/or language.

Since ODSMs are implemented at the higher level, they have less performance but offer more flexibility. It is also system independent [3]. As per the need of the application, programmers can control the distribution of data. Determination of parameters such as, the method of data access, block size, communication mechanism, memory consistency model, etc can be done by programmers easily in ODSMs. There are few subsystems, which may be deployed for the implementation of ODSMs. Examples of such subsystems are TCP/IP, DCE, etc. This may result in the reduction of development time and system becomes portable to variant architectures. Amendments to the concluded system can be easily done once innovative techniques become available [3].

As mentioned earlier in the same section, ODSMs can be implemented at the language/ compiler layer or the library layer. This clearly reveals that new compilers or preprocessors must be developed in order to implement ODSM. Automated manipulation of partitioning the data, distribution of data, parallelism exhibition and optimization in communication are in infancy. For library-layer implementations primitives must be developed which may be useful for programming and compilation. These primitives will be those for data distribution, for the utilization of efficient memory consistency model, and for performance tuning. So it is definitely possible and feasible to implement the DSM in library layer [3].

Object-based DSM has three main advantages:

i. Flexibility in the implementation ensured by controlled data access.
ii. Modularity-the very structure of an object ensures modular organization of data and its related functions.
iii. Transparency -Programmers are unaware of the synchronization of objects and the access done to the objects [7].

The main disadvantages of ODSM are:

1. Shared address space can not be randomized for writing or reading by any process since objects are structurally organized as a block of memory as compared to variables organized as byte-granular.
2. Additional overhead is ensured as accesses to the shared objects must be done by invoking the objects' methods only [7].

It may be possible to apply many types of memory consistency models with no effect to the programming since processes cannot directly access the internal state of a shared object. Generally in ODSM, the *update* or the *invalidation* coherence policy is used [5]. For synchronization generally locks and barriers are used.

Some examples of programming languages for the purposes of implementation of object-based DSM are Linda, Orca [1, 6].

3. INTRODUCTION TO OPDSM- A HYBRID MODEL

In the previous section we discussed, at length, the comparison between Page-based DSM (PDSM) and Object-based DSM (ODSM). As discussed, both have advantages and disadvantages. Having discussed those details, we now introduce our proposed model called Object-and-Page-based DSM (OPDSM) that is a hybrid model, with a mix of advantages and disadvantages.

In the proposed model, the underlying page-based DSM system shown has the same architecture as that of the standard model and is used to transfer pages from one processor node to another in the distributed system. Page-based DSM does not require a programming language runtime support as the page reference management is done by the MMU and the underlying operating system [10].

The hybrid architecture contains an underlying page-based DSM superimposed with an object-based programming language runtime on top of the page-based DSM. The two layer model is as shown in Figure 2.

As can be seen from the diagram, the upper layer is object-based DSM, implemented in the user space, using programming language and its runtime. Any existing ODSM can be used as upper layer of the hybrid model **with few modifications.** The lower layer is implemented in the kernel space of the distributed O.S. Any existing PDSM can be used as the lower layer projected in the hybrid model.

The hybrid model processes the information for page management that includes sending the pages to needy processes that run on distributed nodes, and hence would function in the same way as the ones on traditional DSM systems.

The basic idea of designing this model is to identify the objects as per the application that could be required by one or more processes on a remote node.

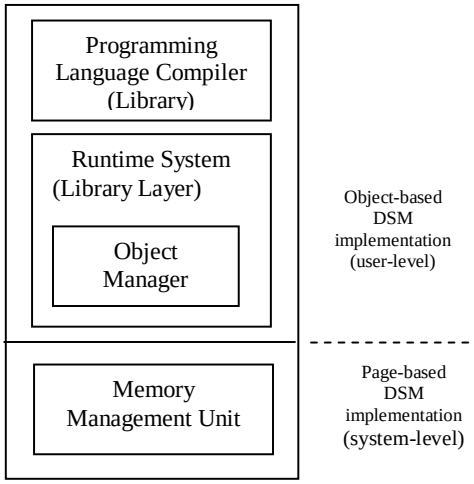

Figure 2. Object-and-Page-based Distributed Shared Memory (OPDSM)

The required data objects can now be sent to the target location by the run time system developed for this purpose. In other cases, again based on applications, objects can be sent over the network to the destination processor in terms of a page using conventional page-based DSM. The hybrid model can adopt any consistency model as that of the object-based DSM system. The consistency model will take into account complete user transparency. In order to perform synchronization without replication, the sequential consistency model can be used in the hybrid model implementation. The sequential consistency model basically performs page invalidations after page updates.

4. IMPLEMENTATION OF OPDSM-HYBRID MODEL

4.1 User-level Implementation

For management of objects, certain run-time system needs to be implemented that should be supported by the programming language to be run on the distributed environment for object sharing. This run-time system has few functions implemented which take care of object manager and all object management.
Every object will be assigned a type, which will indicate the memory model applied.
This type will either be page-based or object-based memory model applied to object. We will discuss the data structures needed for this type implementation in the next section. We assume that at any moment of time there is a single owner processor node of an object, which may reside any where in the distributed system.

The run-time system will have functionalities as specified below.

i. ***int allocate(int size)***
This function allocates the memory for the object, given the size for the object and, returns a unique id (integer) generated for this object.

ii. ***void free (int id)***
This function will be invoked to free the memory of an object if this object has been used and needs to be invalidated or updated.

iii. ***int check _access(int OId)***
This function checks whether an object is sharable i.e. if it is readable or writable. This is done by checking the accessibility field in the Object-look -up table as described in the next section. If this field is 1 then it is writable-and-readable and, if it is 0 then it is readable.
The function returns an integer value that represents the accessibility code.

iv. ***void set_object_info(int OId)***
This function (member of the *ObjectInfo* class)sets the various attribute values that define the specific object. The argument *OId* identifies the object the attributes of which are defined by the class *ObjectInfo* with the following members;

```
class ObjectInfo
{
        int m_iOId; //the object identifier
        bool m_boolObjType;
                //0: page based; 1:object based

        int m_iObjSize; //object size in bytes
        int m_iObjOwnerNodeId;
        //owner node identifier
        struct m_stObjUserNodeProcessList
        {
                int m_iObjUserNodeId; //identifier of
                nodes accessing the //object
                m_iOId
                int m_iProcessId;
                //identifier of processes //accessing the
                object m_iOId
        } stObjUserNodeProcessList[ ];
        bool m_boolAccessibility;
                //read access- 0;
                //read-write access- 1
public:
        ObjectInfo( ObjectInfo obj);
        void set_object_info(int OId);
        void get_object_info(int OId);
        void update_obj_lookup(ObjectInfo   temp);
        void update_p_lookup(ObjectInfo   temp);

}
```

v. ***void update_p_lookup(****ObjectInfo* ***obj)***
This function does the job of updating the Process-wise Object Sharing Information table (Table 1).

vi. ***void update_obj_lookup( )***
This function does the job of updating the Object Distribution table (Table 2).

vii. ***void send (****ObjectInfo* ***ObjI,*** *ObjType* ***Obj,*** *int* ***DestinationNodeId)***
This function is invoked by the runtime system of the owner node to send the object with *OId* (Member of ObjI) to destination node identified by *DestinationNodeId*. The actual object to be shared is of type *ObjType.*

viii. ***void lock (****ObjectInfo* ***ObjI,*** *ObjType* ***Obj)***
This function is used for mutual exclusion.
Only one object at a time can be in its critical section to be writable with the help of this function. Thus this object cannot be copied to any other node when it is in its critical section.

ix. void Unlock (Object OId)

This function is also used for mutual exclusion. The function unlocks the object and takes the object out of critical section and hence can be copied to any processor node.

As mentioned earlier there exists a process called the object manager, that resides on every node which is amenable for executing the above mentioned functions.

5.2Data Structures for Object Management

This section discusses the different data structures for the object management.

Following diagrams show thet two lookup tables that are required for object sharing groups.

4.1.1 Process-wise Object Sharing Information Table (Owned by Object Manager on every node)

As shown in Table 1, the object manager maintains the information regarding the processes running at the node and also offers lookup on the local and remote objects manipulated or accessed by that process.

We now discuss meaning of each field and the intention behind deploying this data structure.

1. ObjUserNodeId: - This field identifies the node on the distributed system assuming that every node in the distributed system is assigned a unique identifier.

2. ProcessId: - This is process id that owns this object. This *ProcessId* indicates that this process currently owns this object indicated by *OId*.

Table 1: Process-wise Object Sharing Information

ObjUserNodeId	ProcessId	OId	Object Size (in bytes) ObjSize	Object Location ObjOwnerNodeId	Accessibility (R-0; R/W-1) boolAccessibility
N1	P1	O1	24	N6	0

Table 2: Object Distribution Information

ObjOwnerNodeId	OId	Object Size (in bytes) ObjSize	ProcessId	Process Location ObjUserNodeId	Accessibility (R-0; R/W-1) boolAccessibility
N1	O2	20	P3	N5	0

3. OId: - Identifier of the object residing on node *ObjUserNodeId [*9].

4. ObjSize: - The object with *OId* has **ObjSize** size in bytes.

5. ObjOwnerNodeId: - The object *OId* is owned by node identified by **ObjOwnerNodeId**

6. Accessibility: - This field indicates the access rights to an object indicated by OId; for eg, this object may be readable(indicator bit : 0), writable(indicator bit : 1)or both (indicator bit: 1).

The above data structure is manipulated by Object Manager process using ***void update_p_lookup( )*** function mentioned earlier in the user-level implementation. All the Object Managers residing on different nodes on DS must update this data structure as and when processes and objects get created and destroyed.

4.1.2 Object Distribution Information Table

The second data structure is as shown in Table 2. As shown this data structure provides the Object Manager process with object-look-up information as compared to process-wise lookup information (as in Table 1).

We now discuss meaning of each field and the intention behind deploying this data structure.

1. ObjOwnerNodeId: - This field represents the node on the distributed system that owns the object (*OId*).

2. OId: - Object with this object id resides on the node *ObjOwnerNodeId*.

3. Object Size: - size of the object OId is specified in bytes.

4. User Node-Process Pair: - The information about the user node location and user process that accesses an object is indicated in two fields of Table 2. The user node-process pair is represented by a structured list of data object, **stObjUserNodeProcessList**, whose members are **ObjUserNodeId and ProcessId** .

The runtime system updates this data structure by using ***void update_o_lookup( )*** function mentioned earlier in the user-level implementation section.

4.3 Kernel-level Implementation

This section deals with the implementation of OPDSM model in kernel level. As mentioned in the section "Introduction to OPDSM- a Hybrid Model" any existing software for Page-based DSM can be considered to be the lower layer of our proposed model.

Since the software PDSM is constructed by amendment in the underlying OS, user level library, runtime system, linker and preprocessor [6], our proposed model does not take care of any PDSM implementation. Instead the user-level implementation as mentioned by ours will suffice the purpose.

4.4WORKING OF OPDSM

Having discussed the two types' implementations and the data structures for the user-level implementation, we now head towards how all the above layers get integrated to form the working model of our system. Let us first have a look at the modified user layer.

Any object will be created on any processor node using the *allocate( )* function by the runtime system. As mentioned earlier in the user-level implementation, type field is associated with every object at the time of creation of object based on type of application. The object will either be page-based object or object-based object. Based on the type of the object *(ObjType)* the runtime system will take decision about further distribution and access.

If the type is object model then the object will be sent using the user-level runtime system. The size of the object is specified when it is created and hence, the ***granularity*** in object based implementation is bound to be an object.

As mentioned earlier in the user-level implementation section there is only one owner of the object. Hence when an object is needed by any processor node it is copied to that node using the *send( )* function. Before being sent, it is checked for accessibility. If it is readable (bit value:0) then the copying can be done and nothing needs to invalidated. If it writable and/or R/W (bit value:1) then this shared object is updated by some processor node. Then the copies of the object at other processor nodes will be invalidated (using the *free()* function). As it reveals, our model uses the invalidation type of ***coherence policy***.

Our model can work for any type of **consistency model** used at the user level. Since only one node can write on an object at any given time, our model uses Many Readers Single Writer (MRSW) **algorithm** [6]. Finally, the functions *lock ( )* and *unlock ( )* are used for **synchronization** (and hence mutual exclusion) of the objects.

The main intention behind the above proposed model and the discussion done in the working so far, is focused on manipulation of objects in an elegant manner by amending the existing ODSM.

Now we discuss the functionality of the lower layer in short, as PDSMs are quite prevalent. If all the objects have their type specified as page based memory model, then all these objects are integrated to form a page size and are sent over network to other processor nodes when needed.

Since size of all objects collectively may not form a page size, internal fragmentation may occur. The rest of the functionality is as per the underlying PDSM, as we have mentioned earlier no amendments have been done to the lower layer of the hybrid OPDSM architecture, since the said lower layer corresponds to the PDSM architecture.

4.5THE FEASIBILITY OF OPDSM

In this section we try to focus on the feasibility study of our model although we cannot be sure about it in practice until this system is actually implemented. Hence we refer to OPDSM as a model and not as a system. We have proposed the model keeping in mind the deficiencies in the implementations of existing ODSMs. Our model tries to mix the pros and cons of the existing PDSM and ODSM. Our main focus was on ODSM as we feel that there is still scope for a lot of improvements in the ODSM. We decided to keep the lower layer as it is and tried to amend only the upper layer with our own speculation.

We feel that our model will result into a system which may have the following advantages and disadvantages.

4.5.1 Advantages

a) Any DSM can be used, since the run time will decide the fate of objects for their transaction from one processor node to another. This will result in the flexibility offered to users.
b) Any consistency model as that in the ODSM can be used.
c) Object access is secured since only the member methods of the object can operate on them.

4.5.2 Disadvantages

a) Integration of existing system has huge overhead although changes need to be done only in the ODSM.
b) Integrated Software DSM like this would have huge memory requirements.
c) Complexity of the system can never be ignored.

5. CONCLUSION

Many existing PDSMs and ODSMs are present in the real world with different features. All these systems have their pros and cons. As mentioned in the earlier section, ODSMs have scope for a lot of improvements.

We have tried to present our proposed hybrid model after comparing the existing PDSM and ODSM. We have tried to put our best efforts to present the model using integration of existing systems and our amendments to one of the systems in an elegant fashion.

6. REFERENCES

1. Andrew S. Tanenbaum Distributed Operating Systems:, 2004
2. George Couloris, Jean Dollimore, Kindberg Tim Distributed Systems: Concepts and Design, 3/e-
3. Adsmith: An Efficient Object-Based Distributed Shared Memory System on PVM-1087-4089/96 $5.00 0 1996 IEEE
4. Distributed Shared Memory Consistency Object-based Model-Journal of Computer Science 3 (1): 57-61, 2007ISSN1549-3636© 2007 Science Publications
5. Ce-Kuen Shieh, An-Chow Lai, Jyh-Chang Ueng Tyng-Yue Liang, Tzu-Chiang Chang, Su-Cheong- Cohesion: An Efficient Distributed Shared Memory System Supporting Multiple Memory Consistency Models Taiwan: 0-8186-7038-Xl95$4 .00 0 1995 IEEE.
6. Jelica Protit, Milo Tomasevit, and Veljko Milutinovi,,DSM: Concepts and Systems- - 1063-6552/96/$4.00 0 1996 IEEE
7. Mordechai Geva and Yair Wiseman -Distributed Shared Memory Integration 1-4244-1500-4/07/$25.00 ©2007 IEEE
8. Htway Htway Hlaing , Thein Theinye , Win Aye -A Simple and Effective Software Distributed Shared Memory System978-1-4244-2101-5/08/$25.00 ©2008 IEEE
9. Abdelfatah Aref Yahya and Rana Mohamad Idrees Bader-Distributed Shared Memory Consistency Object-based Mode,l -Jordan , Journal of Computer Science 3 (1): 57-61, 2007, ISSN1549-3636, © 2007 Science Publications
10. Richard E. Schantz BBN Technologies, Douglas C. Schmidt, Middleware for Distributed Systems Evolving the Common Structure for Network-centric Applications, Irvine, USA