# Adapting the Internet Protocols in a transactional DSM Operating System

**M. Wende, O.Schirpf, M.Schöttner and P.Schulthess**

**Distributed Systems Department, Ulm University, Ulm, Germany**

## ABSTRACT

Plurix is a native cluster Operating System (OS) written in Java. The cluster nodes communicate via Distributed Shared Memory (DSM) and the consistency of the DSM is guaranteed by restartable transactions and optimistic concurrency control. Standard Internet protocols lead to a smooth integration of Plurix nodes into existing network environments and open up the world wide web to Plurix. In this paper we present the design and implementation of network protocols for the management of Plurix transactions and DSM on top of IP. We discuss strategies to support long lived TCP connections in an environment where transactions need to be short to minimize collision probability. Finally we present preliminary performance results of an early prototype which was shown at the CeBIT 2000 trade fair.

**Keywords:** Distributed Shared Memory, Network Protocols, Transactions, Optimistic Concurrency Control, Operating Systems, Plurix.

## 1. INTRODUCTION

Plurix supports the concept of distributed virtual storage [5][6] where distributed memory access is transparent to the application. Remote object access is resolved automatically by the OS. This DSM concept simplifies development of distributed applications because explicit communication as used by Java/RMI, DCOM or CORBA is avoided. Programming of distributed applications appears just like programming to the local machine and the distributed memory is not visible to the application programmer. In a telecooperation scenario for instance users may concurrently edit a document. The document consists of one or more objects residing in the DSM and Plurix transaction preserve the semantic consistency of the text objects.

The Plurix DSM presents a single address space for all cluster members. Objects in the distributed shared memory are directly referenced from individual nodes. Objects are directly accessed even if they are not local. If an application wants to read or write a remote object, for example a shared document, the application only references the object. If this object is not locally available, the MMU causes a pagefault. A pagefault is a hardware interrupt that blocks the application until the requested page is available. Within approximately 500µs the Plurix network protocol fetches the page and makes it available to the application.

To point out the difference between our approach and standard communication mechanisms like CORBA, DCOM or Java/RMI we now briefly discuss traditional distributed object computing.

Distributed object computing extends object oriented programming system by allowing objects to be distributed over a network. Distributed objects can be located on different computers but appear to an application to be local. Traditionally however, objects live in their own client/server address space, whereas in our system there is only one address space for the cluster.

### CORBA

OMG's [10] Common Object Request Broker Architecture relies on the concept of Internet Inter ORB Protocol (IIOP) for accessing remote objects [7]. This protocol uses the Object Request Broker (ORB) acting like a central object bus over which each CORBA object can transparently interact with other CORBA objects. These CORBA objects can be accessed locally or remote. Each CORBA object has an interface exposing a set of methods. If the client wants to access a server object, the client dynamically requests a object reference. The client can now make server object calls like local ones in the client address space. The ORB is responsible for finding a suitable CORBA object implementation, preparing the server object for remote requests and for the communication between client and server object. The client interacts with the ORB through an object adapter. OMG's CORBA is an interface-specification and can be used on different OS platforms if an ORB is implemented on these platforms. Different ORBs can communicate in heterogenous networks using a General Inter-ORB Protocol (GIOP). GIOP is a set of message requests ORBs can make over the network. One standardized GIOP is the IIOP.

IIOP depends on TCP/IP or RPC which allows reliable internet communication. CORBA runs on heterogeneous platforms and offers language independence which is not possible in Java/RMI and in Plurix. Java/RMI and Plurix can only communicate between Java applications. Plurix protocols are designed for communication between PCs running Plurix-OS.

In order to access a CORBA object it is prepared for remote access and registered at the object registry. Parameters for remote calls are marshalled and demarshalled by stub and skeleton routines.

In a DSM system, however, marshalling and demarshalling is avoided. There is no need for costly serialization and deserialization of memory structures because the arguments of a method are loaded on demand. If a parameter is not locally available the Plurix network protocol retrieves the page containing it. It is obvious that there is no need for a object registry or object modification. Because remote objects are referenced like local ones by an application, marshalling and an object registry is unnecessary .

Plurix and CORBA are difficult to compare. In spite of their conceptual difference they are both faced with the problem of concurrent access when objects are modified by different nodes leading to inconsistent versions.

## 2. OBJECT CONSISTENCY

Since memory pages are distributed and replicated in the DSM cluster strict consistency is not available. Using a less constrained consistency model is therefore mandatory.

The consistency model we decided to use relies on optimistic concurrency control (OCC) [1][4]. This strategy prevents deadlocks because all objects are accessible at all times. In case of concurrent writes on an object operations may be rolled back. Optimistic concurrency control occurs in four steps:

1. Resources are copied before they are modified
2. Compare the access pattern of concurrent operations
3. In case of no conflicts: copies are discarded
4. In case of conflicts: operations are rolled back thus undoing all modifications.

OCC requires that operations can be restarted in case of conflict. By keeping transactions short and working sets small we can reduce the probability of a conflict and the overhead arising from roll backs is kept tolerable.

## 3. MICRO TRANSACTIONS

In order to undo operations all reads and writes to objects are part of a micro transactions. Before or during operations the state of the system has to be saved so it can be reestablished in case of roll back.

Our micro transactions comply with the ACID paradigm [11]. Transactions will either complete all actions or will be reset in case of conflict with another micro transaction. A conflict arises if two or more transactions have overlapping read/write sets. This can happen if either one transaction reads a page written by another or if two transactions write to the same page. Of all transactions involved in a collision at least one will be restarted automatically.

At commit time a transaction tries to synchronize to the distributed shared memory and thus to all other nodes. The commit request may fail if an access conflict with another node is detected. In particular n-1 stations out of n are restarted if their write sets overlap.

A variety of schemes can be implemented to minimize the overhead involved in restarting transactions. In any case it is important to keep transactions short and working sets small. A simple collision resolution strategy is "first wins". Commit requests in the cluster may be serialized by using the MAC-protocol available or by an explicit commit token.

### Consistency and Marshalling

In the previous chapters the Plurix DSM concept and the CORBA model for distributed object computing are presented. Both concepts allow concurrent access to distributed objects and invoke extra costs for object consistency. Objects access patterns are compared and operations are encapsulated by transactions. Overhead arises in case of a restart of an operation. The DSM-concept allows to access remote objects like local ones without special efforts like marshalling and demarshalling. Local applications can run in a distributed environment without changes in the code of these applications.

## 4. THE PLURIX PROTOCOLS

### DSMP paging requests

To support DSM consistency and virtual memory management we designed the protocol DSMP. The DSMP data transfer protocol requests pages and packets containing page data. Nodes in need of a page available in the DSM send page request messages. These requests are immediately broadcast to all cluster members and the node owning the requested page answers. The answer packet is unicast to the requesting node.

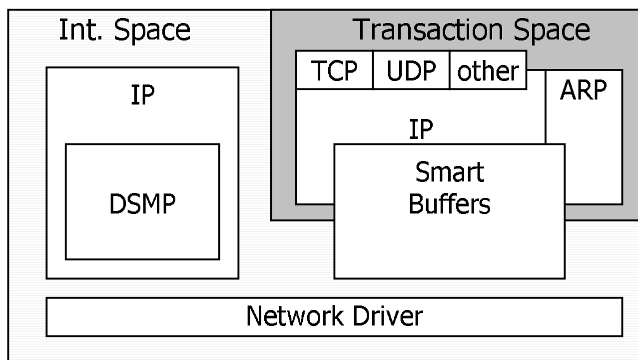


**Fig. 1.** Organization of the protocols

Using these messages each cluster member can access the DSM pages.

A DSMP request packet consist of an IP and a DSMP header. The DSMP header (fig.2) size is 20 bytes: a type of packet (2 bytes), a version number (2 bytes), a heap number (8 bytes) and the logical address of the requested page (8 bytes). The type can be a request or an answer packet The data field of an answer packet contains a copy of the logical page requested.

| Type | Vers | log. Address | heap number |
|---|---|---|---|

**Fig. 2.** DSMP request & answer packet

In the DSM separate but identical copies of the same page can exist at a given time. The concept of optimistic concurrency synchronizes all page replicates. This is achieved by commit messages of the transaction protocol. Commit messages are sent by the transaction part of the DSMP protocol.

**DSMP commit requests**

Nodes requesting pages will receive replicates. After a page request there are replicates on two or more machines. All versions can be read or written by cluster members and therefore a mechanism is needed to avoid version conflicts.

At the end of a transaction the protocol tries to commit all read and write operations to the global memory space. The commit message synchronizes the operations performed during the last transaction to all other nodes. All objects, read or written during the last transaction are compared by broadcasting the addresses of all accessed memory pages. The cluster members detect collisions by comparing received addresses with the accessed ones.

There are three types of transaction DSMP packets:

- sendReadWriteSets: this packet type is used to broadcast addresses of all read or written pages during the last transaction.

Before the read/write set can be sent it must be ensured that only one station sends read/write sets. This is secured by a token protocol as described above.

- sendTokenRequest: this packet type is used to requests the commit token from the last committing node.
- sendTokenGranted: this packet type is used to send a token granted message to the token requesting node.

When the commit token is granted the transaction typically ends successfully by sending read/write sets (the commit message).

Transaction DSMP packets consist of an IP and DSMP header. The DSMP header (fig.3) consists of 20 bytes containing the version number - one of the three types described above - and a commit-timestamp. The timestamp is 8 bytes long and guarantees that packet errors are detected and resolved. The last 8bytes are the heap number of the actual Plurix heap. (In case several DSM heaps coexist.)

| Type | Vers | commit timestamp | heap number |
|---|---|---|---|

**Fig. 3.** DSMP commit packet

The Plurix specific protocol DSMP relies directly on IP. There is no need for special port numbers or secure TCP connections. DSMP doesn't need ports because the protocol is only used by the OS. Applications are not allowed to send or receive DSMP packets.

Page requests and answers need no reliable connection. In case of a packet loss, page requests are sent again. The commit message is protected by the commit number. This number is incremented after each successful transaction. If a packet gets lost during transmission the commit numbers of the following commit packets will be incorrect until the lost commit message is sent again.

## 5. IMPLEMENTATION OF THE PROTOCOLS

The protocol DSMP is handled at interrupt level. A pagefault caused by the MMU is an interrupt that suspends the current operation and sends a *request for page* packet. If a page request is received the network adapter interrupt determines whether to send an answer packet or if the requested page is locally not available. All operations concerning the virtual memory management are initiated by pagefault or network card interrupts. DSMP messages are not encapsulated in transactions because requests for pages may be received at anytime. The commit messages are responsible for the transaction protocol and therefore are excluded from transactions. On this account the DSMP protocol runs operations performed by the protocol in the interrupt space. These operations are not monitored by transactions.

The Plurix protocol stack is illustrated in figure 1. Receiving a packet, the network driver calls higher protocols for further processing. The *interrupt space IP*, the SmartBuffers and ARP are tentatively called until one protocol reads the data out of the received packet. If the received packet is a DSMP packet the *interrupt space IP* finds the DSMP signature in the IP header. If there is no matching signature, the protocol returns and the network driver calls the next protocol. The protocols UDP, TCP, ARP and IP are located in the transaction space. Operations, especially operations of these protocols, are monitored by transaction mechanisms. IP and ARP packets are identified and saved in the smart buffers.

**The smart buffers**

TCP, UDP and other standard protocols work in the transaction space where all operations can be restarted. If a network packet is received during a failing transaction the data must remain available until a successful transaction consumes the packet As described in the consistency chapter pages are only modified after a copy of the original page is saved. During commit the copy is discarded and in case of abort the modified page is replaced by the original one. This mechanism would delete packets that have been received during an aborted transaction.

However, packets received should survive until a committed transaction has read them. This problem is solved by the use of smart buffers. Received packets are written to both versions of the page they are saved in (fig 4). A commit discards the copy and in case of an abort the same data is written back. Read and write pointers, pointing to the actual packet are saved the same way. The write pointer of a smart buffer for incoming packets is written to both page versions. Therefore a following packet could not overwrite the last received packet, not even if the operation is aborted. The read pointer is only once available in the system and is reset by standard mechanisms if an operation is restarted. This guarantees, that restarted transaction reads the same packet again.

Applications using UDP or TCP/IP may abort during execution. On this account packets should not be sent over the network before the transaction ends successfully. The transaction ACID paradigm requires that operations are either executed or completely restarted. If a TCP packet is created during a transaction it is only sent to the smart buffer were it is stored until the transaction commits. After the commit of a transaction the read and write pointers of the smart buffers for outgoing packets are verified and packets in the buffer can be sent.

Smart buffers guarantee that received packets are stored until a successful transaction reads them and that outgoing packets are sent after the commit of a transaction.

**Long lived TCP connections**

TCP connections and UDP packets are needed for communication between Plurix nodes and the non-Plurix world. Processing of a TCP stream might involve many short transactions. It is unavoidable that some of these transactions are aborted due to an external memory conflict. The aborted transaction will then resume and read its packets again form the smart buffer. The TCP connection will remain active independent of the restart of some of its transactions.

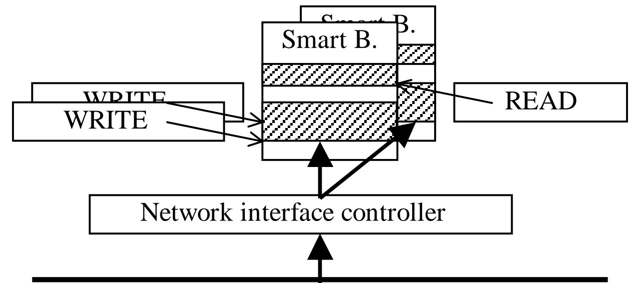


**Fig. 4.** Smart buffer for incoming packets

All received packets are securely stored in smart buffers, and packets created during a transaction are deleted in case of abort before they are physically sent.

However received messages are not removed by an abort and new messages are only sent after commit of a transaction. Consequently long living TCP connections can be established and communication protocols wouldn't be influenced by transactions.

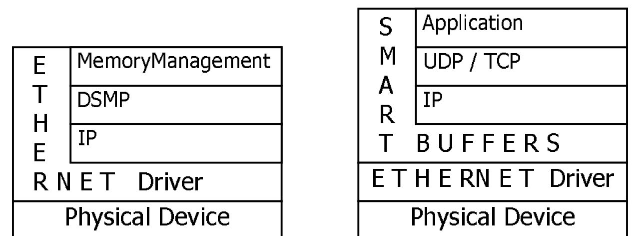


**Fig. 5.** Piece lists

**DSMP and the IP protocol stack**

Using the smart buffers to send and receive TCP or UDP packets causes some overhead because received packets have to be copied to both versions of a page. These mechanisms are necessary for standard protocols but they are not required for Plurix specific protocols. DSMP is running independent from transactions and the messages for DSMP are immediately forwarded. Commit messages have highest priority being responsible for memory consistency of the Plurix cluster. DSM messages: request packets and answers containing copies of pages, are directly moved between memory and network card to

minimize latency. Also commit messages containing read/write sets are sent without intervening smart buffers reducing any overhead.

**Piece lists**

All Plurix network protocols rely on IP (RFC791) allowing smooth integration into a contemporary network environments. Typical network layer models imply copying of data between each layer but this is too costly for DSMP. Therefore our IP implementation adopts the concept of piece lists. References are passed from one network layer to another, saving CPU time.

Fig. 5 presents the way data is moved between different network layers - on the left side for DSMP and on the right for standard protocols. If the memory management sends a page over the network DSMP generates a header. A reference to the header and a reference pointing to the page are transferred to IP. IP creates a header, attaches a pointer to this header and passes all references to the Ethernet driver. References are passed only until the Ethernet driver fetches and sends the data. UDP and TCP packets as well are transferred as references from one layer to another until continuous packets are saved in the smart buffers. The reception of packets is done the same way. Each network layer reads only the relevant part from the receive buffer or the smart buffer respectively.

This implementation of an IP stack using piece lists leads to a fast and lean implementation. Some preliminary results are presented in the next chapter. There are some measurements from our demo at the CeBIT fair and from a special transaction test application.

## 6. PERFORMANCE EVALUATION

Our demo application presented at the CeBIT fair consists of three PCs interconnected by a 100 Mbit/s Ethernet Hub running a video-conference application and a simple GUI. The video conference application shows two remote and one local videos with the image size of 266*200 pixel and a refresh rate of 20 frames per second. The video images consist of raw data with two byte color depth without compression. This leads to 106kBytes per frame. The refresh rate of the videos was limited by the VGA video card not by network protocols or the 100Mbit Ethernet.

| # Plurix nodes | 3 |
|---|---|
| Frames / second | 20 |
| Picture size | 266 * 200 |
| Bits per frame | 106400Bytes *8 |
| resulting Network traffic | 51 MBit/s |

**Fig. 6.** Results of Plurix demo application

Additionally we ran some performance tests using an application running on three machines continuously incrementing one integer in the DSM. This application causes many aborts because the same value is concurrent permanently accessed. The nodes reload the page containing the value after each abort. Using this demo setup we got the following values:

| Total transaction rate (100Mb, >2 Stations) | 6760 |
|---|---|
| Request for page (4KB) | |
| 10Mbit | 5ms |
| 100Mbit | 0.5ms |

**Fig. 7.** Transaction rate and req. for page time

These are preliminary results and more specific tests are scheduled to be performed.

## 7. CONCLUSION AND PERSPECTIVE

This paper presents and evaluates design and implementation of Plurix specific network protocols. Beyond Plurix network protocols we discussed standard protocols like UDP and TCP. These are necessary to support connections with standard OS like Windows NT and the Internet. Strategies concerning long lived connections are presented.

Future work concentrates on testing and revision of Plurix specific and standard network protocols. There is a preliminary version of UDP/IP which is being extended. After the complete implementation of TCP and UDP they will be tested by FTP, SMTP or HTTP applications.

A formal protocol design effort is envisaged which will concentrate on: correctness, security, fairness of collision resolution and simplicity.

## 8. REFERENCES


1. Kung, H.T. and Robinson John.T., *On Optimistic Methods for Concurrency Control* , ACM Transactions on Database Systems 6, 2, 213--226, 1981
2. Traub, Stefan, *Speicherverwaltung und Kollisionsbehandlung in transaktionsbasierten verteilten Betriebssystemen*, Ph.D.Thesis, University of Ulm Germany, 1996
3. Schulthess, Peter and others, *DSM-Java: Foundation of a Lean Distributed Operating System*, Proceedings of the International Workshop on Distributed Computing on the Web, Rostock, Germany, 1999
4. Thomasian, Alexander, *Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing,* IEEE Transactions on Knowledge and Data Engineering, Volume 10, pp. 173--188, 1998

5. J.L. Keedy and D.A. Abramson. *Implementing a large virtual memory in a Distributed Computing System.* In Proceedings of the Hawaii International Conference on System Sciences, 1985.
6. K. Li. IVY: *A Shared Virtual Memory System for Parallel Computing*. In Proceedings of the International Conference on Parallel Processing, 1988.
7. Gopalan Suresh Raj. *A Detailed Comparison of CORBA, DCOM and Java/RMI*, http://www.execpc.com/~gopalan/misc/compare.html
8. Schulthess, Peter and others, *DSM-Communities in the World-Wide Web*, Workshop on Distributed Communities on the Web, Quebec City (Canada), June 2000
9. *Locking and Concurrency* http://www.globis.ethz.ch/education/oodb/objy/guide/jgdLocking.fm.html
10. *OMG* http://www.omg.org/
11. *Distributed Transaction Processing (TP): Parts 1/2/3: Model/ Service/ Protocol*, International Organization for Standardization, Information Technology, Open Systems Interconnection, International Standards 10026-1/2/3, ISO/IEC JTC1/SC21, 1992