UNCLASSIFIED

DEPARTMENT OF DEFENCE

DSTO

Information Technology Division

RESEARCH NOTE
ERL-0719-RN

A FULLY REPLICATED DISTRIBUTED DATABASE SYSTEM

by

S.J. Miller

APPROVED FOR PUBLIC RELEASE

ELECTRONICS RESEARCH LABORATORY

UNCLASSIFIED

AR-008-470

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Information Technology Division

RESEARCH NOTE
ERL-0719-RN

A FULLY REPLICATED DISTRIBUTED DATABASE SYSTEM

by

S.J. Miller

SUMMARY

This paper investigates a fully replicated distributed database system suitable for Naval single platform combat systems. It is found that the fully-distributed approach to update synchronisation, where each site completely executes every update, can be tailored to minimise the control message traffic between sites. Database reliability issues are discussed and a number of possible approaches for maintaining data consistency in a fully replicated database are considered.

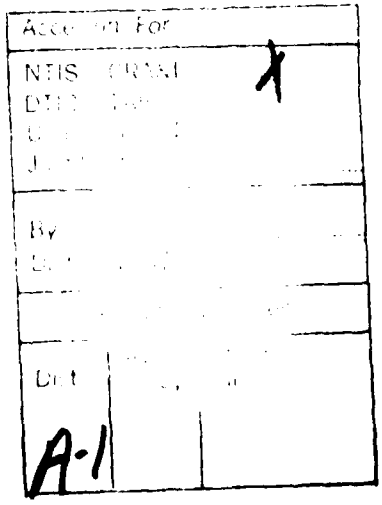
JUNE 93

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1500, Salisbury, South Australia, 5108.

CONTENTS

FIGURES



ABBREVIATIONS

ADDAM	Admiralty Research Establishment Distributed Database Manager
C	Clock value at site i
DDBM	Distributed DataBase Manager
DM	Database Manager
DOQ	Data Output Queue
DRAM	Dynamic Random Access Memory
FDA	Fully Distributed Algorithm
f	Function
LRQ	Local Request Queue
RM	Reliability Manager
RQ	Read data Reply Queue
RRQ	Request Reply Queue
RSQ	ReadSet Queue
rs	Readset
SN	Sequence Number
S_i	Site i
TH	Transaction Handler
TM	Transaction Manager
TP	Transaction Processor
TS(U)	Timestamp of transaction U
t	A clock value
ts	Timestamp
ts1	Updated timestamp
U	Transaction
ws	Writeset
XOQ	External Output Queue
XRQ	External Request Queue

THIS IS A BLANK PAGE

1 INTRODUCTION

During the last decade there have been a number of proposals for single platform Naval Combat Systems using a distributed processing architecture. These systems contain a number of processors loosely connected by a high speed bus or a Local Area Network (LAN). Increased reliability and availability have been achieved through the redundancy provided by this architecture and by distributing the real-time tactical database over the networked processors. Because combat systems are required to process high rate input data in real-time, the database must be stored in main memory. In these early distributed systems, processor main memory was expensive and bulky, resulting in processors with relatively small main memory capacities. As a consequence, databases in these early systems were not fully replicated but, were partitioned and portions replicated at different processor sites.

Increased processor speeds, lower main memory access delays, and cheaper and smaller main memory, now make the implementation of fully replicated real-time distributed database systems for Naval combat systems practical. In the fully replicated database, data objects are redundantly stored at all sites giving maximum reliability and responsiveness to the database system.

The update of distributed and replicated data objects, whether in a fully or partially replicated database system, must be done using concurrency algorithms. These algorithms control the interleaving of conflicting actions so as to maintain the database integrity. Updates are generally expensive in time because they involve extensive communication for synchronisation; however, if a fully distributed method for controlling updates is employed, a significant improvement in performance can be achieved over that possible for a partially replicated database.

Singhal(1) suggests that a way of reducing the overhead and resulting delays of synchronising communications is to use a Fully-Distributed Algorithm (FDA). This algorithm requires significantly less synchronising communication as the actual data object values are not sent to each site with the update. Instead, a "readset" of the data objects required, a function, a data object "writeset", and a timestamp are broadcast to each site. The readset is a list of the data objects required to be read and used as input to the function. The function is applied and the results are written to the database objects listed in the writeset. Processors at each site can perform the required update at its site independently and in parallel with the other sites without the need for a large number of control communication messages being sent between the originating site and the other processor sites. A further improvement in the FDA may be possible in situations where the types of transactions (atomic sequences of updates which may include functions), can be pre-determined during design and built in as known types. The single platform Naval combat system has a well defined functional requirement and would allow suitable tailoring of the FDA to improve its performance in this manner.

Investigations in this paper show that, although the FDA does provide significant performance gains when considering updates on existing database data using other existing database data, these gains are much reduced when the creation of new records, and the updating of existing records, from external data are considered. In such cases, data values do need to be communicated between the distributed sites. Singhal does not address this aspect at all.

This paper investigates the applicability of fully replicated databases and their associated control in real-time distributed processing systems suitable for future single platform Naval combat systems. Section 2 addresses the fully replicated database in terms of the inherent architectural advantages and the physical hardware implications. Section 3 discusses distributed database management update protocols and the improvements in performance obtainable when using the

FDA in conjunction with a fully replicated database system, while section 4 looks at possible specialised tailoring of the FDA to further improve update performance in applications such as combat systems. Section 5 summarises the advantages of the fully replicated database and the relevance of the FDA to Naval combat systems employing this database architecture.

2 FULLY REPLICATED DATABASES

Distributed processing systems incorporating fully replicated databases are inherently more reliable than systems having partitioned and partially replicated databases. Since every site has a complete copy of the database, greater system functional integrity is provided in the event of network partitions. A network partition can occur when a break occurs in the communication medium isolating one or more processor sites. In addition to the greater reliability, there is a reduction in the associated control required. There is no need for the complicated mechanisms for maintaining owners (the primary data copy) and an acceptable number of subscriber copies (secondary copies or replications) of each database partition, as used in the Admiralty Research Establishment U.K. Database Manager (ADDAM) and similar systems(2,3). There is also an improvement in performance obtainable from the use of local copies of data at each site. Faster access memory systems and higher speed communications improve performance of both partially and fully replicated database systems. However, potential improvements in performance can only be achieved by using update algorithms which require significantly less inter-site communication.

An example of a homogeneous distributed processing system architecture incorporating a fully replicated database system is shown in figure 1. This figure depicts a three site distributed system connected by a communication medium. Each site has identical processing hardware and the same software architecture. The data base is fully replicated and managed by the local Distributed DataBase Manager (DDBM).

3 CONCURRENCY CONTROL FOR REPLICATED DATABASE SYSTEMS

Updates of data objects in a replicated environment are expensive as they involve extensive communication for synchronisation. Several algorithms have been developed over the last decade for synchronising updates. Basically, all algorithms can be divided into two fundamental classes. Algorithms which perform synchronisation before accessing data objects, referred to as "pessimistic" algorithms, and those which perform synchronisation after accessing the data objects, referred to as "optimistic" algorithms. Generally, algorithms used for updates in replicated databases adopt one site as the controlling site for performing the synchronisation, executing the update, and finally distributing the new values to other sites. Such algorithms are referred to as "semi-distributed".

Singhal(1) proposes a fully distributed approach and suggests that a fully replicated database system, where each site completely executes the update using an FDA, would have the following features:

a. High parallelism, because after an update is sent to each site, the sites can be execute it in parallel with the other sites.

b. Being fully replicated, there is less reliance on communication among sites; there are no complicated control protocols for implementing commit strategies, and hence is a more reliable algorithm.

c. No need to transfer data object values between sites and hence shorter messages are required.

d. Higher data security and privacy as there is no exchange of data values.

e. Greater update performance, as there is no waiting required for the computing of values and writing of values at other sites, as each site performs the update in its entirety.

The above features lead to a system having improved performance, greater reliability and lower communication overhead. This increased performance was evident in performance studies of the FDA(1). The study confirmed that not only is the communication overhead cut down, but there is also an enhanced performance gain due to the faster access available to main memory data objects.

It should be pointed out that Singhal's study only addressed the clear cut comparison between the two situations of having and not having data object values communicated between sites and not a combination of both. Singhal does not consider the need to communicate actual data values in practical distributed database systems. In a real system, data must be input into the database from an external source and this data must be distributed to other sites. Further, at initialisation, when a site joins a distributed network, or rejoins after a network partition or site failure, there has to be a data exchange to make the database replications consistent.

In spite of these apparent oversights, there does appear to be benefits to be gained from using a modified form of the FDA in fully replicated database systems. Its tailoring for use in the Naval combat system application is addressed in section 4.

3.1 Description of FDA

A full description of the FDA will not be given here as this is available in reference 1. Instead, only a sufficient description will be given to provide a basic understanding of its protocol, its application interface and inter-site message requirement, to provide a base for understanding how it can be tailored to an application like the Naval combat system.

A user or application modifies a database by submitting an update at its local site in the network. An update U is defined(1) by a three-tuple U = (rs,ws,f) where:

a. "rs" indicates the data objects read by U and referred to as the "readset".

b. "ws" indicates the data objects that are written to by U and referred to as the "writeset".

c. "f" is a function which models the computation required and results in new values f(rs) for the writeset.

An update U, when executed alone, is required to take the database from one consistent state into another. When two or more updates co-exist and where conflict exists between them, an inconsistent database can result. The FDA provides concurrency control using timestamps and a set of synchronisation rules to prevent these inconsistencies occurring.

The timestamp of an update is unique across the entire system. Each site S_i has a logical clock C_i, which takes on a monotonically increasing integer value(4). When a user submits an update at a site S_i, C_i is incremented by one and a two-tuple (C_i,i) is assigned to U. This is referred to as the timestamp TS(U) of U. Every message contains the current clock value of its sender site, and when a site S_j receives a message with clock value t ($t = C_i$), it sets the local clock value C_j to $\max(t+1,C_j)$.

When a site receives an update from a user, it forms an update message containing a timestamp and sends it to all other sites. Each site performs the update completely using local database data. That is, each site looks after synchronisation and executes the read, compute, and write actions. The update execution is performed in a fully distributed manner. The synchronisation is embedded into the FDA.

3.1.1 FDA embedded synchronisation

The FDA performs read and write synchronisation(1). A site executes the read action of an update U when:

a. the site has received a message with a timestamp larger than TS(U) from every other site, received all updates up to timestamp TS(U) (ie there are no earlier updates yet to be delivered from external nodes), and

b. write action of all updates having their timestamps smaller than TS(U) and having write-read (w-r) conflict with U have been executed at that site.

Similarly, a site executes the write action of an update U when:

(a) read action of all updates having their timestamps smaller than TS(U) and having read-write (r-w) conflict with U have been executed at that site, and

(b) write action on all updates having their timestamps smaller than TS(U) and having write-write (w-w) conflict with U have been executed at the site.

3.1.2 The FDA process

When a site S_i receives an update U = (rs,ws,f) from a user, it takes the following sequence of actions(1):

(a) It updates its clock and assigns a timestamp, say "ts" to U.

(b) It sends an UPDATE(rs,ws,f,ts) request message to all other sites and saves the update along with its timestamp.

When a site S_j receives an update request from site S_i it takes the following actions:

(a) Site S_j responds to the update request by first updating its clock,

(b) returns a REPLY(ts1) (ts1 is the updated timestamp) to S_i, and

(c) it saves (rs,ws,f,ts).

Site S_i increments its clock from the reply (see section 3.1) and all sites execute the update in accordance with the synchronisation rules in section 3.1.1.

3.1.3 FDA proof of correctness

A formal proof of the FDA correctness is provided in reference 1. Singhal shows that internal and mutual consistencies of a replicated database are maintained despite concurrent update execution and that every update does execute in a finite time. Singhal also proves that the FDA is free from deadlocks and updates will not wait indefinitely in a circular chain.

3.2 Effects of site and media failures

Sites and communication media are prone to failures. These failures may result in lost messages and network partitions. Singhal(1) mentions a number of possible strategies to handle these failure situations. The actual strategy will be driven by the degree of reliability required. Reliability can be built into transaction processing by employing mechanisms which timeout if reply messages are not received from all database replications, and then retry the transaction a number of times before finally aborting the transaction.

Update reliability, and hence database consistency, is often traded for availability in combat systems(ref. 2). That is, in the eventuality of a network partition or node failure, the database system is maintained in a useable condition with some degradation of performance and loss of data consistency. In these systems, the update transactions will not abort; instead, they will be allowed to complete at the originating site even without replies from all the remote sites. Remote sites perform the externally requested update immediately in accordance with the synchronisation rules. They do not need to wait for a further message from the originator to proceed with the update. A "crash" algorithm will need to be implemented at the originating site so that when unavailable sites recover or the network recovers, recovery algorithms can be run to bring all the database replications into a globally consistent state.

3.2.1 Loss of messages

One method of handling the loss of messages is to use a timeout mechanism. When a site S_i sends an update message to S_j, it initiates a timer. If the timer expires before the return of the reply messages from all S_j then, S_i again sends the update. When S_j receives the update request message, it responds to it, and all subsequent duplicates of it, with a reply message. However, it requires a unique identifier to be sent with both the update and reply messages to ensure they can be matched together. If a site S_j receives a duplicate message it will reply without performing the update process. Depending on the degree of reliability, this process can continue until a reply has been received from all sites or, if the emphasis is on availability, continued for a predetermined number of times, or a timeout can be employed. A journal (time ordered log of transactions) of updates missed by any given site S_j (no replies), can be kept and used to restore global consistency when communication is again possible with that site. This journal will need to be replicated across all the available sites to ensure it is not lost because of a site or network failure. The complexity of maintaining journals and their application in the recovery process is addressed in a separate paper(8).

3.2.2 Site failure

A site failure can also result in a message being lost. In this case there will be no reply from the failed site. This will either block the update at the sending site from being performed, or if the availability approach has been adopted then, after a given number of tries, the update will continue, the update request being kept in a journal for the failed site to proceed on its repair. Of course long down times require large journal storage. An alternative is for the failed site, on repair, to request a new copy of the database to re-initialise itself. An incremental recovery strategy(7,8) would be used to achieve this.

3.2.3 Network partition

A network is partitioned when divided into two or more network groups by a severing of the media or by interface hardware failures. Each group can work independently if the availability strategy is adopted; however, no update is possible between them. This situation is the most complex, as updates could occur on different copies of the same database, and so on repair it would be necessary to take into consideration all the updates from all portions and merge them together. In this case journals must be kept on each partition and when repair occurs, all inconsistencies must be resolved to produce a globally consistent database from the individual journals. The precedence graph method(5) or the log transformation technique(6) can be used to do this.

4 TAILORING THE FDA TO THE COMBAT SYSTEM APPLICATION

Each application makes certain demands on a database. The Naval single platform combat system is required to perform a set of defined functions including sensor data collection, data processing and evaluation, tracking, target assessment, threat analysis, weapon allocation and engagement. All these functions require database access for compilation, calculations and data presentation or display purposes. Further, the data must be processed and acted upon within the time constraints and deadlines imposed by the current threat.

As the combat system is required to process large numbers of contacts and tracks derived from different input sensors in a similar manner, track data and contact data are stored in similar record formats and can be accessed by means of a contact (CN) number or track number (TN). Further, since many of the operations to be performed on each track are the same and involve data previously updated, the required operations can be defined and implemented by a transaction sequence. It is then only necessary for an application to provide a TN and the transaction type, instead of the data object readset, function, and writeset, when requesting an update. In essence, there are two basic types of transactions involving updates, those concerned with inputting new data and requiring the communication of data between sites, and those essentially operating on existing database data which can be performed in parallel at each site without any data transfer.

There are two further cases, requesting for a database copy or a local read, where the exchange of data between sites or between the local database and the application is also required. At initialisation when a node first joins or rejoins the network and during periodic database consistency checking, if employed, an exchange of data is necessary. The "copy_request" transaction is useful in these cases. A copy_request message includes the database partition or record, depending on the database granularity required, and a timestamp. If a number of "copy_request" messages are required to initialise a site's database, then it may be necessary to lock the relevant database partitions until each transaction has completed. A copy_request updates the local database from

the relevant reply data. The special case of the local read transaction does not require data values be exchanged between sites. However, it does return data from the local database to the application.

A further distinguishing feature of transactions relates to how they are to be performed. They may be performed in a "reliable" fashion, or by a "performance" process. A "reliable" transaction is one in which the copies of the appropriate data objects in all replications, or some pre-determined minimum number of them, must be updated or, if this is not possible then none of them is to be changed (see section 3.2). An example is the case when data relating to a contact classification, is inserted by an operator. In the case of high input data rate sensor type data, it is not so important if the odd input is missed as new data will soon be available; any database inconsistencies will be only temporary and will be eventually rectified. In this situation a reliable process can be traded for a performance one(2,3).

The FDA synchronisation protocol can be used for transactions involving reliable updates even when data must be communicated between sites. In order to provide the required serialisation of updates to provide concurrency, a timestamp system is often used, as is the case with FDA. Each time an application requests any transaction, it increments its local clock counter C_i and formats a transaction request message containing the transaction type and an updated timestamp $ts = (t,i)$ where $t = C_i$ (section 3.1).

Transaction processing and database control can be divided between two processing modules. The transaction message handling, decoding and reliability enforcement can be performed by a Transaction Manager (TM) while the database control, including concurrency using the FDA, and the simple performance update, can be handled by a Database Manager (DM).

The FDA protocol integrity is not compromised in this application. The protocol is implemented with the minor addition of some handshaking in the form of acknowledgments from each external node when a reliable process is required. This addition was included to deal with the typical real world communication system which may not be error free or fault tolerant. Handshaking is not used for the performance update. The following sections examine this approach in greater detail.

4.1 Transaction Manager

Transactions are processed by the TM. Typically, a TM could have the functionality shown in figure 2, consisting of a Transaction Handler (TH), a Transaction Processor (TP), a Reliability Manager (RM), and a number of interface queues. The TM interfaces with the application, the database and the communication interface. All interfaces are by means of message queues.

The application is interfaced by a Local Request Queue (LRQ), and a read data Reply Queue (RQ), while the TM interfaces to the communication interface through the External Request Queue (XRQ), the external Request Reply Queue (RRQ), and the External Output Queue (XOQ). Three extra interface queues are provided to the DM. These are the ReadSet Queue (RSQ), the WriteSet Queue (WSQ), and the Data Output Queue (DOQ).

Communication with the TM queues is by means of messages containing a timestamp and identifying information. These messages may also include data object values as in the special transaction types of copy_request and read. The identifying information could be transaction type, writeset data objects, readset data objects, or reply data, depending on the queue applicable. Each group of identifying information will also include a sequence number for matching requests with replies.

4.1.1 Transaction types

Transactions are groups of database accesses, associated computations, and database changes, or combinations of them, which must be performed as a single unit or process. To ensure that the database remains consistent, transactions must be serialised so that updates from one transaction do not conflict with those from a second transaction. A transaction must be able to be totally performed or not performed at all. In a typical real-time database application such as a Naval combat system, a set of transactions can be defined at system design, and hence built into the system as pre-defined types. These may be called by an application process simply by their name type and by supplying the data object set involved. A sample list of typical transaction types is defined in Appendix I.

4.1.2 Transaction Handler

The TH accepts messages from the external and local queues in timestamp order. External messages are handed over to the TP for decoding and further processing. Locally initiated transaction messages are accepted and passed to the RM to hold for local processing when the reliability requirements have been met. Only the originating site for the transaction addresses the reliability issues.

4.1.3 Transaction Processor

The TP further decodes and processes the external transaction messages passed to it from the TH and the local transaction messages passed on from the RM, according to their types. Each time an external transaction is accepted the local clock is set to $\max(C_i,t+1)$ and the sequence number is checked to determine if it is a repeat of a previously accepted transaction. If it is a repeat of a previous transaction, no local action is taken except, a reply message including an updated timestamp and the sequence number is formatted and placed on the XOQ for posting to the requesting site. Otherwise, the reply message is generated as above and appropriate actions are taken. In the special case, when the external transaction is a copy_request, the reply is delayed until the required data has been obtained from the DM. The read data is formed into an external output reply message and placed on the XOQ for the communication interface to send to the requesting site. For both local and external transaction messages, readsets are generated and sent to the local DM together with the appropriate timestamp and a sequence number. Readset replies from the DM are sorted according to type and sequence number.

When the readset reply is associated with an update transaction requiring specified calculations, the calculations are performed and the appropriate writeset data object values generated, formatted into a message including the transaction timestamp and posted on the WSQ for the DM. In the special case, "update contact", where the data values are also included with the read set, then the TP does not perform any calculations but simply updates the writeset values with them.

If the readset reply is for a remote copy_request transaction, the local clock C_i is set to $\max(C_j,t+1)$ (C_j is obtained from the external timestamp and t is the current value of C_i), and the read data is formatted into a message together with an updated timestamp (C_i,i) and posted to the requesting site by placing it on the XOQ. The requesting site is derived from the timestamp (C_j,j) accompanying the readset reply. Finally, if the readset

reply is in response to a Read type local transaction, the read data is formatted into a message and placed on the RQ for the application

When a reply is received to a local copy_request, the reply data is converted into a writeset and posted on the WSQ for the DM.

4.1.4 Reliability Manager

The RM must ensure the reliability and availability characteristics required for the database are met. It implements such measures as timeouts for reliable transactions sent to other sites and initiates a repeat sending of the transaction copy (broadcast), or it passes the transaction over to the TP for processing. Background consistency checking measures could also be implemented. In the special case of a performance transaction (update contact), no reply messages are expected or looked for and the transaction is passed immediately over to the TP.

For each local transaction request requiring the reliable protocol, the RM formats a broadcast message, places it on the XOQ, and then waits for transaction replies on RRQ from each site, or until a set timeout. Each time a reply is received, the local clock is set to max(C_i,t+1) (see section 3.1). An update reply is of the form REPLY(ts,SN) where ts is the timestamp and SN a Sequence Number. If the reply is as the result of a copy_request, it will include the read data (REPLY(ts,data,SN)). When all associated replies have been received, or at timeout and predetermined availability criteria have been met, the transaction request is passed on to the TP for processing. In the case of a copy_request the request and the reply data is passed on to the TP.

If replies are not received from all the replications, possibly due to a lost message or site/network failure, then a new copy of the broadcast message is placed on the XOQ. This procedure will be repeated a pre-determined number of times and eventually, if the required number of site replies are not received, will timeout and an appropriate crash algorithm will be implemented to retain a transaction log for the apparently unavailable sites(8). The crash algorithm will need to ensure the log is not unique and therefore vulnerable in the case of a site failure, see section 3.2.1.

4.2 Database Manager

The DM processes the messages from the RSQ and WSQ queues containing the readsets and writesets values according to the FDA synchronisation rules, section 3. Readset messages which meet the synchronisation rules are executed and their values are formatted into messages and sent back to the TP through the DOQ. Writesets and associated values obtained from the writeset queues which meet the synchronisation rules, are used to update the local database.

4.3 Site recovery

The recovery process for a site in a network having a fully replicated database system is similar to when a new site joins the network. A recovering site must first update its database to match the global database. To do this, the recovering site must issue a sequence of "Copy_Requests" (transaction), one for every record or page, depending on the database granularity, for which updates have occurred during the failure. In the case of a new site, a complete copy of the database is required. Copy_Requests must be synchronised at the external site by concurrency control algorithms just as with database update transaction

processes. After the first Copy_Request has been issued by a recovering site, all the following transactions issued over the network must be serialised (logged in timestamp order) at the recovering site. No data item is to be read on the recovering site until it has been written to by a Copy_Request. This could be achieved by the use of "guardians"(7), see section 4.3.3. After the local database is updated and the recovery process is complete, all logged transactions can be processed in accordance with the synchronisation rules. There are a number of ways in which the copy_request process can be applied. Two methods are as follows:

(a) Broadcast the Copy_Requests and accept the first reply.

(b) Post the Copy_Requests to the nearest numbered site in the global network.

4.3.1 Broadcast method

Site recovery using the broadcast method is a heavy user of communications. It involves broadcasting a sequence of Copy_Requests to all sites in the global network and processing the first reply received. Each site receiving the requests, provided it is not itself recovering, reads its local database copy, formats a sequence of replies containing the portions of the database requested, and posts it to the requesting site. The first sequence of replies reaching the requester is accepted and used to update its local database. All subsequent reply sequences are ignored.

4.3.2 Nearest Site method

A simpler approach to site recovery is to keep track of the available nodes in a network partition using a simple site availability algorithm, and request a copy of the data from the nearest available site. A site is considered available if it is active and not currently in the process of recovery. The recovery process requires the recovering site to first perform an availability algorithm to obtain local knowledge of the available sites, and then post a transaction sequence of Copy_Requests to the nearest site, either the next lowest or next highest numbered available site determined by a simple algorithm. Very little overhead would be required to keep an updated list of available sites.

4.3.3 Guardian approach

The guardian approach(7), can be used to ensure a logical data item at a recovering site is not updated until it has been written to by the recovery process. For each logical data item at the new site, designate a "guardian copy" at an old site. After the new site is added, the site holding the guardian copy alerts all transactions that updates should write into the new copy also.

4.4 Network repair for partitions

Network partitions are much more difficult to recover from than site failures. If two partitions have been operating independently from each other, and there has not been any restrictions placed on the way database objects could be updated at either partition, then the databases will have diverged. On repair, the two databases need to be merged together in some way taking into account the time sequence of events in both halves (section 3.2.3).

The network repair situation can be reduced to simply multi-site recovery if certain restrictions are placed on the updating of data objects during a partition. If each data object is given a token site on the non-partitioned network then when a partitioning occurs, the updating of data objects can be restricted to the partition holding the token site for that data object. Such a scheme however, is not always practical as several sites often require update privileges on the same object set. Further, if the token site fails, the object will not be able to be updated. An alternative token site cannot be selected because of the uncertainty whether the token site has actually failed or a network failure has left it in a separate partition.

The subject of maintaining fully replicated distributed databases in the presence of network or node failure and repair is considered to be a topic in its own right and will be covered in another paper(8).

In practice the likelihood of network partitions can be minimised in combat systems by adopting a dual ring LAN such as the Fiber Distribution Data Interface (FDDI) (see ref 9).

FDDI has two relevant techniques for improving combat system network reliability:

(a) A station bypass switch to bypass failed or powered down stations.

(b) Counter rotating ring connections where the second ring may be used for standby and when a link fails the two rings can be folded into one double length ring to maintain Connectivity.

Further, by paying careful attention to the physical placement of the fibre medium, and the use of duplicated and triplicated bus solutions, the possibility of action damage can be minimised.

5 CONCLUSION

Recent technological advances in microprocessor performance, together with much greater main memory capacity at reduced cost and reduction in memory access times, has led to a revival of interest in distributed processing systems with fully replicated real-time databases. In real-time applications, such as Naval combat systems, the fully replicated database requires simpler database control mechanisms. In addition it does not need complicated control mechanisms for the maintenance of replicated partitions or to dynamically distribute them around the distributed processing network in the events of network partitions, site failures and on repair. Greater update performance is also available from the fully replicated database systems.

Singhal has suggested that concurrency control can be more efficiently performed in fully replicated database systems and has developed an algorithm (FDA) to provide the necessary synchronisation for updates. The FDA as proposed does not require data values to be transferred between sites, results in much less message transfer and provides a significant update performance gain. However, in real life practical databases, such as used in combat systems, there is a requirement for data to be input to them from the outside world (eg new contact data from sensors). In these cases it is not practical or desirable to interface all input sources to all sites. Thus to maintain the replicated database when new data is entered, data values will have to be communicated between sites. Further, at initialisation of the network or when a site is being introduced or reintroduced after a site failure, all replications of the database must be made consistent and will need an update process which also involves data value transfer.

This paper has outlined a method where the FDA synchronisation protocol can be incorporated into a database manager, and transaction types can be decoded and managed by a transaction manager in a way which maximises the benefits obtainable from the FDA when updates are on existing database data. It also accommodates the requirements for value transfer during updates involving new data inputs and site initialisation.

Site and network partition recovery have also been addressed. The simpler method of obtaining a database copy from the nearest neighbouring site has been put forward as being the most efficient and least expensive of communications for site recovery. The problem of recovery after a network partition is much more complex and is not fully investigated in this paper. This subject needs further investigation and is to be covered in a later paper.

REFERENCES

No.	Author	Title
1	Singhal, M.	"Update Transport: A New Technique For Update Synchronisation in Replicated Database Systems". IEEE Transactions on Software Engineering, Vol. 16, No. 12, Dec 1990
2	Miller, S.J.	"Distributed Processing Test-Bed System: First Interim Report for the DPTBS". WSRL-TM-26/90, Sep 1990
3	Reilly, M. and Tilman, P.R.	"Maintaining Consistency and Accommodating Network Repair in a Self-Regenerative Distributed Database Management System". AGARD-CP-360, Feb 1985
4	Lamport, L.	"Time Clocks and Ordering of Events in Distributed Systems". Communications, ACM Jul 1978
5	Davidson, S.B.	"Optimism and Concurrency in Partitioned Database Systems". ACM Transactions Database Systems, Sep 1984
6	Blaustein, P.A. Garcia-Molina, H.	"Maintaining replicated Databases even in Presence of Network Partitions". EASCON, 1983
7	Attar, R. Bernstein, P.A.	"Site Initialisation, Recovery, and Backup in a Distributed Database System". Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, Nov 1982
8	Miller, S.J. O'Dea, D	"Maintaining Fully Replicated Distributed Databases in the Presence of Network or Node Failure and Repair". To be published.
9	Ross, F.E.	"FDDI - a Tutorial" IEEE, Communications Magazine, May 1986, Vol. 24, No. 5

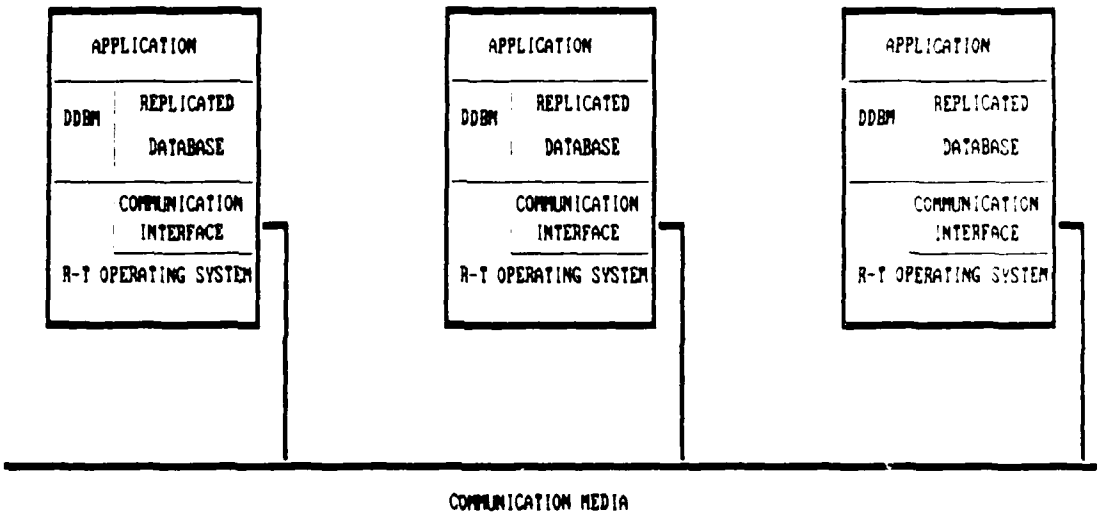


Figure 1 A Fully Replicated Database Architecture

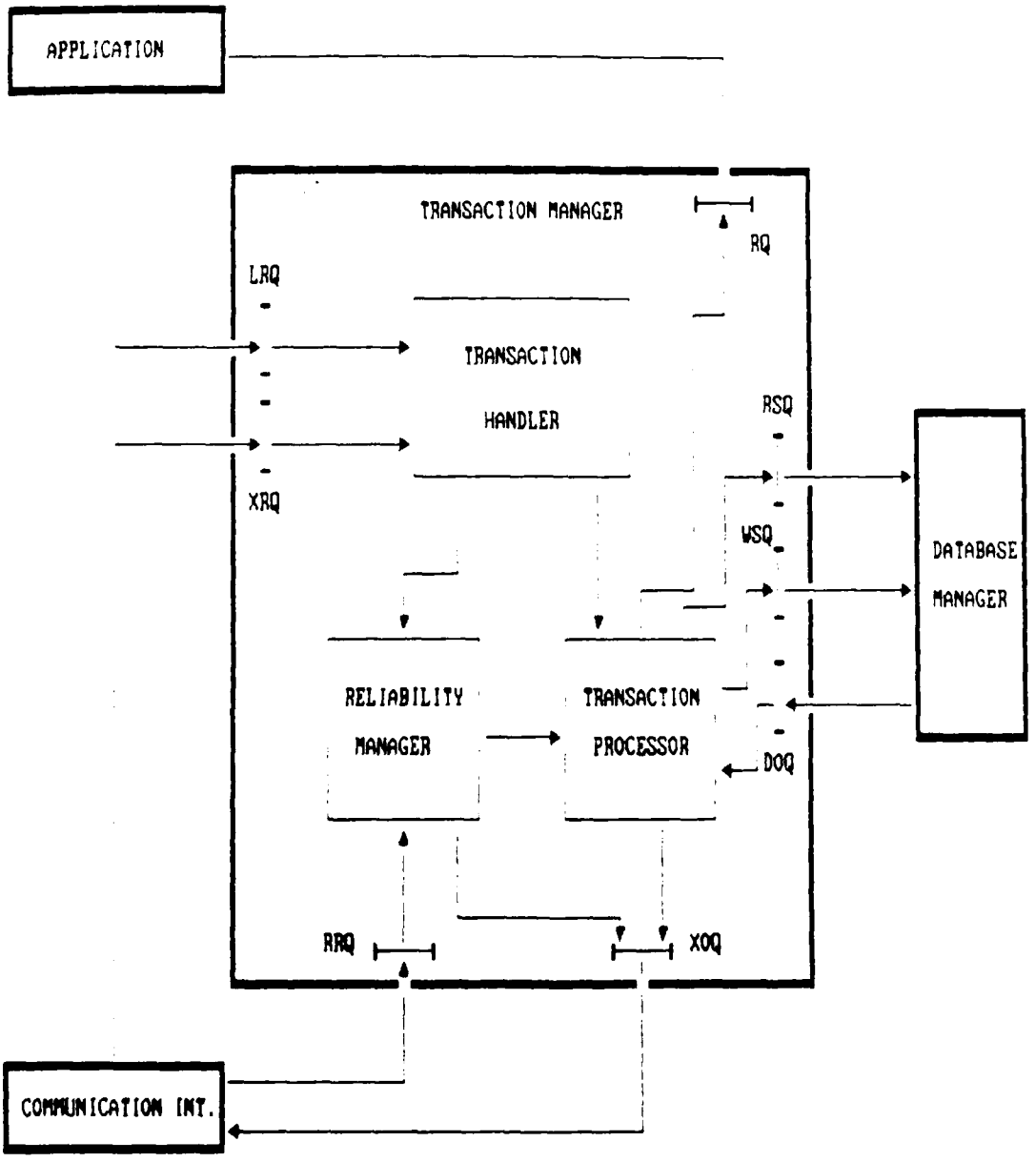


Figure 2 Transaction Manager

APPENDIX I

SAMPLE OF TYPICAL TRANSACTION DEFINITIONS

This appendix defines a representative set of database files and a group of typical transactions which interact with these files.

I.1 Database Files

The following database files types are typical of a combat system:

(a) SENSOR_CONTACT

(b) TRACK_POSITION

(c) TRACK_HISTORY

(d) TRACK_SUPPLEMENTARY

(e) TARGET_DATA

I.2 Transaction Definitions

A typical set of transaction types suitable for combat systems are defined in terms of a call statement including any inputs (IN) parameters required, and output (OUT) parameters obtained by the relevant processing, in the following sections.

I.2.1 New Contact

(a) Call statement

NEW_CONTACT(IN:SENSOR;OUT:CN,ERROR_CODE)

where; SENSOR is an identifier for the associated sensor.
CN the allocated contact number, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-

0 = OK
1 = SENSOR does not exist
2 = Contact file full
3 = error in generating the replications
4 = reliable process error

(b) Transaction processing

Obtain the next available contact number CN and create a sensor contact data file for the sensor type SENSOR. Store the CN and SENSOR identification in the sensor

contact file. This must be a reliable transaction process to ensure that all the required replications are generated. Return the result of this process in ERROR_CODE.

I.2.2 Update Contact

(a) Call statement

UPDATE_CONTACT(IN:CN,SENSOR_DATA;OUT:ERROR_CODE)

where; SENSOR_DATA contains the new data values
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track CN does not exist
2 = SENSOR undefined

(b) Transaction processing

Update track CN with new data from SENSOR_DATA. A performance update protocol can be used. Return the result of the process in ERROR_CODE. This transaction requires the new SENSOR_DATA values to accompany the readset.

I.2.3 Read Contact

(a) Call statement

READ_CONTACT(IN:CN;OUT:CONTACT_DATA,ERROR_CODE)

where; CONTACT_DATA is the read data returned, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track CN does not exist

(b) Transaction processing

Reads the local contact CN file and returns the CONTACT_DATA. Return the result of the process in ERROR_CODE.

I.2.4 Delete Contact

(a) Call statement

DELETE_CONTACT(IN:CN;OUT:ERROR_CODE)

where; ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-

0 = OK
1 = contact CN does not exist
2 = system track exists
3 = reliable process error

(b) Transaction processing

Delete using a reliable process (essential that all copies be deleted) the contact CN data file, including all its replications, if it is not associated with a system track. Return the result of this process in ERROR_CODE.

I.2.5 New Track

(a) Call statement

NEW_TRACK(OUT:TN,ERROR_CODE)

where; TN is the allocated track number, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-

0 = Ok
1 = track file full
2 = error in generating the replications
3 = reliable process error

(b) Transaction processing

Obtain the next available track number TN. Create all the relevant track, history and supplementary records. This must be a reliable transaction process to ensure that all the required replications are generated. Return the result of this process in ERROR_CODE.

I.2.6 Update Track Position

(a) Call statement

UPDATE_TRACK_POSITION(IN:TN,CN;OUT:ERROR_CODE)

where; ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-

0 = OK
1 = track TN does not exist
2 = contact CN does not exist
3 = reliable update error

(b) Transaction processing

Update track TN with SENSOR_DATA from CN. The process will involve the updating of the history records, calculation of track vector and present track position parameters, and the updating of the relevant track record. A reliable update process should be used. Return the result of the process in ERROR_CODE.

I.2.7 Update Track Supplementary Data

(a) Call statement

UPDATE_TRACK_SUPPLEMENTARY(IN:TN,DATA_TYPE,DATA,
OUT:ERROR_CODE)

where: DATA_TYPE is data like Classification, Threat Designation etc.
DATA contains the new values, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track TN does not exist
2 = DATA_TYPE does not exist
3 = DATA address does not exist
4 = reliable update error

(b) Transaction processing

Update track TN supplementary record with new data from DATA of DATA_TYPE. A reliable update process should be used. Return the result of the process in ERROR_CODE. This transaction sends the new DATA values with the readset.

I.2.8 Read Track Position

(a) Call statement

READ_TRACK_POSITION(IN:TN;OUT:TRACK_POSITION,ERROR_CODE)

where: TRACK_POSITION is the returned data, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track TN does not exist

(b) Transaction processing

READ_TRACK_POSITION reads the track TN position data from the local file and returns it in TRACK_POSITION. It also returns the result of the process in ERROR_CODE.

I.2.9 Read Track Supplementary Data

(a) Call statement

READ_TRACK_SUPPLEMENTARY(IN: TN;
OUT:TRACK_SUPPLEMENTARY,
ERROR_CODE)

where; TRACK_SUPPLEMENTARY is the returned data, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track TN does not exist

(b) Transaction processing

READ_TRACK_SUPPLEMENTARY reads the track TN supplementary data from the local file and returns it in TRACK_SUPPLEMENTARY. It also returns the result of the process in ERROR_CODE.

I.2.10 Delete Track

(a) Call statement

DELETE_TRACK(IN:TN;OUT:ERROR_CODE)

where; ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-
0 = OK
1 = track TN does not exist
2 = target track exists
3 = reliable process error

(b) Transaction processing

Delete using a reliable process all the associated track TN data records, including all its replications, if it has not been designated as a target. Return the result of this process in ERROR_CODE.

I.2.11 Copy Request

(a) Call statement

COPY_REQUEST(IN:PARTITION;OUT:DATA,ERROR_CODE)

where: PARTITION is a database portion identifier eg page,
DATA is the read data, and
ERROR_CODE is a code returned to indicate the success/failure of the operation.

Code :-

0 = OK
1 = PARTITION does not exist
2 = reliable process error

(b) Transaction processing

COPY_REQUEST requests a copy of a database PARTITION from an appropriate remote site. The data read at the selected site is returned to the application at the requesting site in DATA. It also returns the result of this process in ERROR_CODE.

DISTRIBUTION

	Copy No.
Defence Science and Technology Organisation	
Chief Defence Scientist)	
Central Office Executive)	1 shared copy
Counsellor, Defence Science, London	Cnt Sht
Counsellor, Defence Science, Washington	Cnt Sht
Scientific Adviser to Defence Central	1 copy
Electronics Research Laboratory	
Director	1 copy
Chief, Communications Division	1 copy
Chief, Electronic Warfare Division	1 copy
Chief, Information Technology Division	1 copy
Research Leader, Command & Control and Intelligence Systems	1 copy
Research Leader, Human Computer Interaction Laboratory	1 copy
Research Leader, Military Computer Systems	1 copy
Head, C3I Systems Engineering Group	1 copy
Mr S. Miller, C3I Systems Engineering Group	2 copies
Mr J. Schapel, C3I Systems Engineering Group	1 copy
Mr A. Allwright, C3I Systems Engineering Group	1 copy
Mr D. O'Dea, C3I Systems Engineering Group	1 copy
Head, Program and Executive Support	1 copy
Head, Trusted Computer Systems Group	1 copy
Head, Software Engineering Group	1 copy
Head, Computer Systems Architecture Group	1 copy
Head, Command Support Systems Group	1 copy
Head, Image Information Group	1 copy
Head, Information Acquisition and Processing Group	1 copy
Head, Information Management Group	1 copy
Head, Systems Simulation and Assessment Group	1 copy
Head, Exercise Analysis Group	1 copy
Publications and Publicity, Information Technology Division	1 copy
Graphics and Documentation Support	1 copy
Department of Defence	
Director General, Communications and Information Systems	1 copy
Defence Intelligence Organisation	
Scientific Adviser - Defence Intelligence Organisation	1 copy
Mr P. Drewer, PRS Intelligence Development and Systems	1 copy
Army Office	
Scientific Adviser - Army	1 copy
Navy Office	
Director, Naval Combat Systems Engineering	1 copy
Naval Scientific Adviser	1 copy

Air Force Office	
Air Force Scientific Adviser	1 copy
Libraries and Information Services	
Australian Government Publishing Service	1 copy
OIC, Technical Reports Centre, Defence Central Library, Campbell Park	1 copy
Manager, Document Exchange Centre, Defence Information Services	1 copy
National Technical Information Service United States	2 copies
Defence Research Information Centre, United Kingdom	2 copies
Director Scientific Information Services, Canada	1 copy
Ministry of Defence, New Zealand	1 copy
National Library of Australia	1 copy
Defence Science and Technology Organisation Salisbury, Research Library	2 copies
Librarian Defence Signals Directorate Melbourne	1 copy
British Library Document Supply Centre	1 copy
Spares	
Defence Science and Technology Organisation Salisbury, Research Library	6 copies

Department of Defence

DOCUMENT CONTROL DATA SHEET

Page Classification
Unclassified
Privacy Marking/Caveat (of Document)
N/A

1a. AR Number	1b. Establishment Number	2. Document Date	3. Task Number
AR-008-470	ERL-0719-RN	JUNE 93	NAV87/226

4. Title

A FULLY REPLICATED DISTRIBUTED DATABASE SYSTEM

5. Security Classification

Document	Title	Abstract
U	U	U

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L) in document box.

6. No. of Pages	29
7. No. of Refs.	9

8. Author(s)

S.J. Miller

9. Downgrading/Delimiting Instructions

N/A

10a. Corporate Author and Address

Electronics Research Laboratory
PO Box 1500
SALISBURY SA 5108

10b. Task Sponsor

NAVY

11. Officer/Position responsible for

Security..... N/A

Downgrading..... N/A

Approval for Release..... DERL

12. Secondary Distribution of this Document

Approved for Public Release

Any enquiries outside stated limitations should be referred through DSTIC, Defence Information Services, Department of Defence, Anzac Park West, Canberra, ACT 2600.

13a. Deliberate Announcement

No limitation

13b. Casual Announcement (for citation in other documents)

[X] No Limitation

[ ] Ref. by Author , Doc No. and date only.

14. DEFTEST Descriptors

Combat systems
Distributed database systems
Real time operations
Naval applications*

15. DISCAT Subject Codes

1207, 1506

16. Abstract

This paper investigates a fully replicated distributed database system suitable for Naval single platform combat systems. It is found that the fully-distributed approach to update synchronisation, where each site completely executes every update, can be tailored to minimise the control message traffic between sites. Database reliability issues are discussed and a number of possible approaches for maintaining data consistency in a fully replicated database are considered.

Page Classification
Unclassified

16. Abstract (CONT.)

17. Imprint

Electronics Research Laboratory
PO Box 1500
SALISBURY SA 5108

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
ERL-0719-RN		RESEARCH NOTE

21. Computer Programs Used

N/A

22. Establishment File Reference(s)

N/A

23. Additional information (if required)

Doc. Sect. WF11