METHOD AND APPARATUS FOR DATA PROCESSING

Field of the Invention

The present invention relates to a method and apparatus for data processing. More particularly, but not exclusively, the invention relates to a method and apparatus for database replication.

Background of the Invention

In a number of data processing applications fragments of data sent from a source system must be processed into a required data format on a target system.

In many instances it is desired to replicate a database on a target computer system from a database on a source system. This process may involve sending journal entries from the source database to allow updating of the target database. Databases may consist of one or more library, each of which contains one or more files, each file having one or more members. Each member consists of a table having one or more rows. A journal entry may contain an identifier of the library; file; file member and a row of changed data for the file member. This journal entry may be used by the target computer system to update its database.

It is important that database entries from a given table are updated in the correct sequence and that inter-related members are updated in the correct sequence. To ensure that journal entries are properly processed a receive process of the target computer system may compare an object name (library/file/member) with a database of objects stored on the target computer system. When a matching

object is located the processing information associated with that object may be used to process the journal entry.

The traditional approach has been to transfer journal entries, store them and replicate the database utilising a single engine. This approach is slow and complex.

It would be desirable for a database replication system to meet the following requirements:

1. Ensure that journal entries are serialised by database member (at a minimum), and by any user specified groupings.

2. Support an extremely large number of database apply processes so that database I/O (input/output) can be easily managed.

3. Process journal entries in a way which minimises the amount of system I/O (e.g. paging) between the time the entries are obtained from the journal and the time it is applied to the replica database.

4. The functions support any type of data packets, not just journal entries, to allow for future extensions to other types of replication (e.g. object, stream files etc).

5. The system hides the complexity of the memory management functions from other components.

It is an object of the present invention to provide a method and apparatus for information replication which meets these requirements or to at least provide the public with a useful choice.

Disclosure of the Invention

According to a first aspect of the invention there is provided a method of replicating information from a source system at a target system comprising the steps of:

i) receiving information strings from the source system; and

ii) assigning the information strings to serialisation groups for processing such that inter-related information strings are processed in the same serialisation group.

The information strings may be journal entries from a source database which may be allocated to serialisation groups so that journal entries of the same type, or which are related to other journal entries, are processed in the same serialisation group.

According to a further aspect of the invention there is provided a method of managing memory space in a data transfer operation comprising the steps of:

i/ defining a plurality of storage space areas;

ii/ writing data to a first storage space area; and

iii/ reading data from the first storage space area whilst no data is written to the first storage space area.

The method may enable multiple simultaneous reads from other data storage space areas whilst information is written to only the first storage space area.

According to a further aspect of the invention there is provided a method of replicating a database from a source computer system at a target computer system comprising:

i) receiving journal entries from the source computer system;

ii) checking the journal entries to see if an entry exists in a dynamic index giving processing information relating to a database member to which the journal entry relates; and

iii) if an entry exists in the dynamic table, processing the journal entry according to the associated processing information; or

iv) if an entry does not exist in the dynamic index, looking up the related processing information for the database member in an assignment database, creating an entry and storing it in the dynamic index; and processing the journal entry according to the processing information.

The entry in the dynamic index may provide information as to whether the member needs to be processed with other members. A journal entry may be temporarily stored before being processed according to the processing information.

According to a further aspect of the invention there is provided a method of replicating a database from a source computer system to a target computer system comprising:

i) receiving journal entries from the source computer system; and

ii) allocating program components to process journal entries and update the target database, wherein a control program allocates tasks to program components and controls the program components substantially without program components interacting with one another.

Preferably the target computer system is a multi-processor computer system.

Brief Description of the drawings

The invention will now be described by way of example with reference to the accompanying drawings in which:

Figure 1: shows a schematic diagram of a source computer system which provides journal entries to a target computer system.

Figure 2: is a functional diagram illustrating the processes involved in database replication at a target computer system.

Figure 3: shows the mapping of storage space within the target computer system.

Figure 4: shows a flow diagram illustrating the process for allocating journal entries to serialisation groups.

Detailed Description of the Preferred Embodiment

The following description describes a database replication method where the source and target computer systems are IBM AS/400

computers operating under the OS/400 operating system. It will be appreciated that the method is applicable to other systems with appropriate modification.

Referring to figure 1, source system A contains a primary database 1. Primary database 1 may contain one or more library. Each library may contain one or more file. Each file may contain one or more members. Each member comprises a table having one or more rows. A unique library/file/member combination is referred to as an object.

When a row of any member of primary database 1 is modified a journal entry including the object name and the modified row is sent to local journal receiver 2. Local journal receiver 2 sends the journal entry via communications link 3 to a remote journal receiver 4 of a target computer system B. A database replication process 5 receives the journal entries and modifies the contents of replica database 6 to maintain it in conformity with primary database 1.

Referring now to figure 2 the process and apparatus for replicating target database 6 of the target computer system will be described. To ensure proper replication of replica database 6, database members are updated in the replica database 6 in the same order as they are modified in the primary database 1. To achieve this a number of serialisation groups 8 are defined. Journal entries having the same object name are grouped into a common serialisation group so that they are updated in the correct order. Certain database members may have relationships with other database members (joins etc) and so may be assigned to a common serialisation group to ensure all inter-related members are updated in the correct sequence. A serialisation group may thus contain journal entries for a number of objects. The use of such serialisation groups enables database replication to be conducted in the appropriate sequence as well as facilitating efficient parallel processing.

Receive process 7 may either assign a received journal entry to a serialisation group, assign a journal entry to a default serialisation group or discard the journal entry. Serialisation group assignment is performed based upon an assignment database (MXSGMBAS) and a temporary OS/400 user index object. The journal entry assignment functions are provided via an ILE service program – which allows the underlying implementation to be modified without recompile/bind of the calling functions.

The assignment database MXSGMBAS contains all objects, their relationship with other objects (i.e. do they need to be grouped with other objects during processing) and their required manner of processing. Assignment of a journal entry to a serialisation group 8 could be conducted simply by comparing the object name of each received journal entry with the assignment database MXSGMBAS and assigning the journal entry to a serialisation group based upon the associated information. However, the assignment database MXSGMBAS contains many objects and considerable processing time is required to perform a database locate operation and extract the relevant processing information. According to the invention a member assignment (MBIX) index temporary object is used to store processing information for an object. This is an index of objects giving their associated serialisation group and related processing information (including a link to their associated control structures).

Referring now to figures 2 and 4 the serialisation group assignment will be described. When a journal entry is received in step 11 receive process 7 conducts a comparison in step 12 to see whether the object is present in the MBIX index. If so, operation proceeds to step 13 and a serialisation group number and database file index (DBFIDX) is returned and processing continues within the assigned serialisation group.

If the object name is not stored in the MBIX index then a full object name lookup is conducted in the MXSGMBAS database 9 in step 14. If the lookup is successful then a serialisation group is returned, a Database File Index (DBFIDX) is assigned which will point to the processing information stored in a dynamic array maintained by the associated serialisation group and an entry is added to the MBIX index in step 15. Each Database File Index (DBFIDX) is created simply by incrementing an index that is unique by serialisation group.

If a match is not achieved in step 14 then a generic name lookup is conducted in step 16. This involves a search by a library/file /*all and then by library/*all/*all. If a generic match is achieved the full name is added to the MBIX table in step 17 and processing continues in steps 15 and 13 as before. If no match can be achieved the journal entry is discarded in step 18.

Accordingly, at startup, there will be no entries in the MBIX index 10. As journal entries are processed, serialisation groups and the processing information for objects will be added to MBIX index 10. The serialisation group and processing information may be much more rapidly obtained from MBIX table 10 than from MXSGMBAS database 9

This method gives the following, significant, performance benefits:

1. The serialisation groups do not need to search for a member's related processing information. They simply maintain the processing information in a dynamic array with the Database File Index as the means of access.

2. All operations relating to a particular member name may refer to the serialisation group and Database File Index value to uniquely identify the member (a "handle").

Referring now to figure 3 the method of memory management within the target computer system will be described. Storage object space is divided up into a number of storage units SU_1-SU_N. Each storage unit has a storage unit header 20. The storage unit header 20 gives the number of serialisation groups which have journal entries in the storage unit. Each data segment consists of a storage entry header 21 and a storage entry 22. Storage entries are aligned on 16 byte boundaries with padding blocks 23 filling any space between an entry and a 16 byte boundary.

Journal entries are passed on from receive process 7 for storage in a storage object space 24. The journal entries from receive process 7 are stored in storage space object 24 in blocks 22. Each journal entry 22 has an associated storage entry header 21 (or handle) which contains information as to the displacement to the next journal entry in the storage unit for that serialisation group and an associated Database File Index (DBFIDX) containing the processing information for the member associated with the journal entry. The processing information is maintained in dynamic memory with the Database File Index as the means of access.

In normal operation journal entries are consecutively written to one storage unit until it is filled and then journal entries are written to the next available storage unit. Once writing to a storage unit has been completed journal entries may be read from the populated storage unit. Partially filled storage units may be read out when system resources are not being otherwise utilised (i.e. no incoming journal entries need to be stored).

This approach means that memory locks are not required during reading and writing. During the writing process the receive process 7 has exclusive access to write to a storage unit. No locks are required during read operations and so journal entries may be simultaneously read to their associated serialisation group. The only locking required is to decrement the value held in storage unit header 20 when the last journal entry for a serialisation group is read out.

The available storage units queue (ASUQ) 25 controls the order in which free storage units are utilised. ASUQ 25 includes a last in first out (LIFO) buffer which stores addresses of free storage units. Journal entries of a serialisation group are read out of a storage unit until a null value is found in a storage entry header. As each storage entry 22 is read out the storage unit header 20 is decremented. When all journal entries are read out completely from a storage unit the storage unit header 20 will be decremented to zero and the storage unit number is returned to the ASUQ and is the first storage unit re-assigned when new journal entries must be written into storage space. In this way the most recently used storage units are maintained active to reduce the working set of storage units to a minimum.

When all journal entries in a storage unit have been read and the storage unit is released the entire address range of the storage unit may be purged without requiring writing of data to auxiliary storage.

Referring again to Figure 2 the manner of processing will be further described. Control process 19 oversees the replication process and controls processing in the receive process 7 and within the serialisation groups 8. In this manner processing can be conducted within each serialisation group without regard to processing within another serialisation group. By having the whole process controlled by an overarching control process 19 each serialisation group can conduct its

processing in isolation without regard to the complexity of the overall operation.

As each serialisation group receives journal entries for a member in sequence the updating of that member in the replica database 6 is sequential also. By processing linked members in a particular serialisation group processing is streamlined.

When a replica database 6 is to be made a primary database partially applied commits must be removed. Firstly, the control process 19 suspends receive process 7 and processing by serialisation groups 8. Control process 19 then identifies all "open" commit groups (e.g. commit IDs that have not yet received a commit or roll back journal entry). These are processed, serially, from the most recent (i.e. the commit group that has the most recent journal entry) to the oldest as follows:

i) a receive process of receive process 7 receives the commit group's journal entries from journal receiver 26;

ii) all entries are assigned to a "default" serialisation group;

iii) the entries are stored in storage unit 24 in the usual manner but are linked in reverse order (i.e. the head of the list is the last entry in the storage unit, with links moving backward until the first entry in the storage unit);

iv) if a storage unit is filled before that commit group's entries are complete, the storage unit is pushed onto LIFO queue TLQ 27 (instead of releasing it to the default serialisation group). Then a new storage unit is allocated (as normal) and entries continue to be stored;

v) when the commit group's available journal entries are completely received and stored in storage units, the storage

units are dispatched to the default serialisation groups in LIFO order. The result being that the serialisation group receives the journal entries in reverse order (from most recent to oldest);

vi) the default serialisation group processes the entries as "reverse" entries (the entries include a flag to indicate that they are "reverse" entries). This results in all inserts being processed as deletes, updates being removed to their prior image and deletes being inserted etc. Only journal entries which had already been applied (e.g. during normal processing) to the database are processed;

vii) the default serialisation group does not perform a commit on the "reverse" entries until it receives the "data commit group" journal entry. This ensures that if a failure is encountered during the "clean-up" the database is in a known state. This enables the "clean-up" to be restarted.

Once all of the "open" commit groups have been "removed" the control process 19 suspends the other processes and the replica database is ready to be used as the primary database.

This method allows rapid "clean-up" of partially applied commits which does not require processing capability of the system to be utilised unless a secondary database does in fact have to be made a primary database.

The method and apparatus of the invention provide a number of advantages as follows:

1. The allocation of storage unit blocks within a storage space object and control of read/writes avoids the need for locks and read/write concurrency issues.

2. The use of serialisation groups enables members to be updated in a serial manner and for inter-related members to be updated in correct chronology. Serialisation groups enable multiple streams of journal entries to be simultaneously processed whilst processing interrelated members together.

3. The use of the MBIX index greatly reduces lookup time for each journal entry. The use of storage entry headers 21 (handles) enables the next journal entry of a serialisation group to be located rapidly.

4. The use of a control process to oversee the operation of the receive process and processing within serialisation groups enables the sub-processes to process information efficiently without the need to interact with other processes.

5. Simple handling of commits where secondary database is to be made primary database.

Where in the foregoing description reference has been made to integers or components having known equivalents then such equivalents are herein incorporated as if individually set forth.

Although this invention has been described by way of example it is to be appreciated that improvements and/or modifications may be made thereto without departing from the scope or spirit of the present invention.

CLAIMS:

1. A method of replicating information from a source system at a target system comprising the steps of:
   i) receiving information strings from the source system; and
   ii) assigning the information strings to serialisation groups for processing such that interrelated information strings are processed in the same serialisation group.

2. A method as claimed in claim 1 wherein the information strings are journal entries from a source database.

3. A method as claimed in claim 2 wherein the journal entries are allocated to serialisation groups such that journal entries of the same type or related to other journal entries are processed in the same serialisation group.

4. A method as claimed in claim 2 or claim 3 wherein, when each database object is first encountered in a processing operation a temporary index object is generated containing the serialisation group and related processing information for that database object.

5. A method as claimed in claim 4 wherein, if a temporary index object already exists the serialisation group for a database object is assigned based upon the temporary index object data.

6. A method as claimed in claim 4 wherein if a temporary index object does not exist a full object name look-up of an assignment database is conducted, the serialisation group is obtained from the assignment database and a temporary index object is generated.

7. A method as claimed in any one of the proceeding claims wherein processing information for a database object is maintained in dynamic memory.

8. A method of managing memory space in a data transfer operation comprising the steps of:

i) defining a plurality of storage space areas;

ii) writing data to a first storage space area; and

iii) reading data from the first storage space area whilst no data is written to the first storage space area.

9. A method as claimed in claim 8 wherein the data is database journal entries.

10. A method as claimed in claim 9 wherein the journal entries are consecutively written to the first storage space area until the first storage space area is at least substantially filled and then data is written to the next storage space area.

11. A method as claimed in claim 9 or claim 10 wherein each journal entry has an associated header that records the displacement to the next journal entry having the same serialisation group.

12. A method as claimed in any one of claims 8 to 11 wherein data is written to a second storage space whilst data is read from the first storage space.

13. A method as claimed in any one of claims 8 to 12 wherein each storage space has a storage space header which records the

number of serialisation groups having journal entries within that storage space.

14. A method as claimed in claim 13 wherein when all data is read out of a storage space, the storage space header is decremented and the storage space is made available for data to be written to.

15. A method of replicating a database from a source computer system at a target computer system comprising:

i) receiving journal entries from the source computer system;

ii) checking the journal entries to see if any entry exists in a dynamic index giving processing information relating to a database member to which the journal entry relates; and

iii) if an entry exists in the dynamic table, processing the journal entry according to the associated processing information; or

iv) if an entry does not exist in the dynamic index, looking up the related processing information for the member in an assignment database; creating an entry and storing it in the dynamic index; and processing the journal entry according to the processing information.

16. A method as claimed in claim 15 wherein the dynamic index includes information as to whether a database member needs to be processed with one or more other database member.

17. A method as claimed in claim 15 or claim 16 wherein journal entries are temporarily stored before the replication process is performed.

18. A method as claimed in any one of claims 15 to 17 wherein the memory management method of any one of claims 8 to 14 is employed.

19. A method as claimed in any one of claims 15 to 18 wherein the replication method of any one of claims 1 to 7 is employed.

20. A method of replicating a database from a source computer system to a target computer system comprising:

i) receiving journal entries from the source computer system; and

ii) allocating program components to process journal entries and update the target database, wherein a control program allocates tasks to program components and controls the program components substantially without program components interacting with one another.

21. A method as claimed in claim 20 wherein the target computer system implements parallel processing.

22. A method as claimed in claim 20 or claim 21 wherein the target computer system is a multi-processsor computer.

23. A computer system programmed to operate according to the method of any one of the preceding claims.

24. A computer program adapted to carry out the method of any one of claims 1 to 22.

25. A computer readable medium containing a computer program as claimed in claim 24.

26. A database replicated by the method of any one of claims 1 to 22.