(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 162 538 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
12.05.2004 Bulletin 2004/20

(51) Int Cl.[7]: **G06F 11/14**, G06F 9/46, G06F 12/08

(21) Application number: **01119226.7**

(22) Date of filing: **12.02.1999**

(54) **Transferring a resource from a first cache to a second cache**

Übertragung von Betriebsmitteln von einem Cache zu einem zweiten Cache

Transfert d'une ressource d'une première mémoire cache vers une seconde mémoire cache

(84) Designated Contracting States:
DE FR GB NL

(30) Priority: **13.02.1998 US 74587 P**
24.11.1998 US 199120

(43) Date of publication of application:
12.12.2001 Bulletin 2001/50

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
99906927.1 / 1 055 173

(73) Proprietor: **ORACLE CORPORATION**
Redwood Shores, CA 94065 (US)

(72) Inventors:
- **Bamford, Roger J.**
  San Francisco, CA 94109 (US)
- **Klots, Boris**
  Belmont, CA 94002 (US)

(74) Representative: **Viering, Jentschura & Partner**
Postfach 22 14 43
80504 München (DE)

(56) References cited:
EP-A- 0 471 282
EP-A- 0 657 813
US-A- 5 327 556

- **AHMED R E ET AL: "Cache-aided rollback error recovery (CARER) algorithm for shared-memory multiprocessor systems" INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING. (FTCS). NEWCASTLE UPON TYNE, JUNE 26 - 28, 1990, INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING SYSTEMS. (FTCS), LOS ALAMITOS, IEEE COMP. SOC. PRESS, US, vol. SYMP. 20, 26 June 1990 (1990-06-26), pages 82-88, XP010019527 ISBN: 0-8186-2051-X**
- **KERMARREC A-M ET AL: "A RECOVERABLE DISTRIBUTED SHARED MEMORY INTEGRATING COHERENCE AND RECOVERABILITY" 25TH. INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING. DIGEST OF PAPERS. PASADENA, JUNE 27 - 30, 1995, INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING, LOS ALAMITOS, IEEE COMP. SOC. PRESS, US, vol. SYMP. 25, 27 June 1995 (1995-06-27), pages 289-298, XP000597800 ISBN: 0-7803-2965-1**

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION

[0001] The present invention relates to techniques for reducing the penalty associated with one node requesting data from a data store when the most recent version of the requested data resides in the cache of another node.

BACKGROUND OF THE INVENTION

[0002] To improve scalability, some database systems permit more than one database server (each running separately) to concurrently access shared storage such as stored on disk media. Each database server has a cache for caching shared resources, such as disk blocks. Such systems are referred to herein as parallel server systems.

[0003] One problem associated with parallel server systems is the potential for what are referred to as "pings". A ping occurs when the version of a resource that resides in the cache of one server must be supplied to the cache of a different server. Thus, a ping occurs when, after a database server A modifies resource x in its cache, and database server B requires resource x for modification. Database servers A and B would typically run on different nodes, but in some cases might run on the same node.

[0004] One approach to handling pings is referred to herein as the "disk intervention" approach. The disk intervention approach uses a disk as intermediary storage to transfer the latest version of the resource between two caches. Thus, in the example given above, the disk intervention approach requires database server 1 to write its cache version of resource x to disk, and for database server 2 to retrieve this version from disk into its cache. The disk intervention approach's reliance on two disk I/Os per inter-server transfer of a resource limits the scalability of parallel server systems. Specifically, the disk I/Os required to handle a ping are relatively expensive and time consuming, and the more database servers that are added to the system, the higher the number of pings.

[0005] However, the disk intervention approach does provide for relatively efficient recovery from single database server failures, in that such recovery only needs to apply the recovery (redo) log of the failed database server. Applying the redo log of the failed database server ensures that all of the committed changes that transactions on the failed database server made to the resources in the cache of the failed server are recovered. The use of redo logs during recovery are described in detail in U.S. Patent Application No. 08/784,611, entitled "CACHING DATA IN RECOVERABLE OBJECTS", filed on January, 21, 1997.

[0006] Parallel server systems that employ the disk intervention approach typically use a protocol in which all global arbitration regarding resource access and modifications is performed by a Distributed Lock Manager (DLM). The operation of an exemplary DLM is described in detail in U.S. Patent Application Number 08/669,689, entitled "METHOD AND APPARATUS FOR LOCK CACHING", filed on June 24, 1996.

[0007] In typical Distributed Lock Manager systems, information pertaining to any given resource is stored in a lock object that corresponds to the resource. Each lock object is stored in the memory of a single node. The lock manager that resides on the node on which a lock object is stored is referred to as the Master of that lock object and the resource it covers.

[0008] In systems that employ the disk intervention approach to handling pings, pings tend to involve the DLM in a variety of lock-related communications. Specifically, when a database server (the "requesting server") needs to access a resource, the database server checks to see whether it has the desired resource locked in the appropriate mode: either shared in case of a read, or exclusive in case of a write. If the requesting database server does not have the desired resource locked in the right mode, or does not have any lock on the resource, then the requesting server sends a request to the Master for the resource to acquire the lock in specified mode.

[0009] The request made by the requesting database server may conflict with the current state of the resource (e.g. there could be another database server which currently holds an exclusive lock on the resource). If there is no conflict, the Master for the resource grants the lock and registers the grant. In case of a conflict, the Master of the resource initiates a conflict resolution protocol. The Master of the resource instructs the database server that holds the conflicting lock (the "Holder") to downgrade its lock to a lower compatible mode.

[0010] Unfortunately, if the Holder (e.g. database server A) currently has an updated ("dirty") version of the desired resource in its cache, it cannot immediately downgrade its lock. In order to do downgrade its lock, database server A goes through what is referred to as a "hard ping" protocol. According to the hard ping protocol, database server A forces the redo log associated with the update to be written to disk, writes the resource to disk, downgrades its lock and notifies the Master that database server A is done. Upon receiving the notification, the Master registers the lock grant and notifies the requesting server that the requested lock has been granted. At this point, the requesting server B reads the resource into its cache from disk.

[0011] As described above, the disk intervention approach does not allow a resource that has been updated by one database server (a "dirty resource") to be directly shipped to another database server. Such direct shipment is rendered unfeasible due to recovery related problems. For example, assume that a resource is modified at database server A, and then is shipped directly to database server B. At database server B, the re-

source is also modified and then shipped back to database server A. At database server A, the resource is modified a third time. Assume also that each server stores all redo logs to disk before sending the resource to another server to allow the recipient to depend on prior changes.

[0012] After the third update, assume that database server A dies. The log of database server A contains records of modifications to the resource with a hole. Specifically, server A's log does not include those modifications which were done by database server B. Rather, the modifications made by server B are stored in the database server B's log. At this point, to recover the resource, the two logs must be merged before being applied. This log merge operation, if implemented, would require time and resources proportional to the total number of database servers, including those that did not fail.

[0013] The disk intervention approach mentioned above avoids the problem associated with merging recovery logs after a failure, but penalizes the performance of steady state parallel server systems in favor of simple and efficient recovery. The direct shipment approach avoids the overhead associated with the disk intervention approach, but involves complex and nonscalable recovery operations in case of failures.

[0014] The document US-A-5,327,556 teaches a fast technique for transferring units of data between transaction systems in a shared disk environment, whereby dirty pages are transferred from an owning system to a requesting system without writing it to disk.

[0015] The document AHMED R. E. ET AL.: "Cache-Aided Roll-Back Error Recovery (CARER) Algorithms for Shared-Memory Multiprocessor Systems", INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING (FTCS), LOS ALAMITOS, US, June 26-28, 1990, IEEE COMP. SOC. PRESS, vol. 20, pages 82-88, 26 June 1990, teaches various cache-aided roll-back recovery techniques for use in shared-memory multiprocessor systems.

[0016] Based on the foregoing, it is clearly desirable to provide a system and method for reducing the overhead associated with a ping without severely increasing the complexity or duration of recovery operations.

SUMMARY OF THE INVENTION

[0017] According to the invention, which is defined in detail in the appended independent claims 1, 22 and 23, a method, apparatus as well as a computer-readable medium carrying computer executable instructions are provided for transferring a resource from the cache of one database server to the cache of another database server without first writing the resource to disk. When a database server (Requestor) desires to modify a resource, the Requestor asks for the current version of the resource. The database server that has the current version (Holder) directly ships the current version to the Requestor. Upon shipping the version, the Holder loses permission to modify the resource, but continues to retain a copy of the resource in memory. When the retained version of the resource, or a later version thereof, is written to disk, the Holder can discard the retained version of the resource. Otherwise, the Holder does not discard the retained version. In the case of a server failure, the prior copies of all resources with modifications in the failed server's redo log are used, as necessary, as starting points for applying the failed server's redo log. Using this technique, single-server failures (the most common form of failure) are recovered without having to merge the recovery logs of the various database servers that had access to the resource.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements and in which:

Figure 1 is a block diagram illustrating cache to cache transfers of the most recent versions of resources;

Figure 2 is a flowchart illustrating steps for transmitting a resource from one cache to another without disk intervention according to an embodiment of the invention;

Figure 3 is a flowchart illustrating steps for releasing past images of resources, according to an embodiment of the invention;

Figure 4 is a flowchart illustrating steps for recovering after a single database server failure according to an embodiment of the invention;

Figure 5 is a block diagram illustrating a checkpoint cycle according to an embodiment of the invention; and

Figure 6 is a block diagram of a computer system on which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0019] A method and apparatus for reducing the overhead associated with a ping is described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other database servers, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

FUNCTIONAL OVERVIEW

[0020] According to one aspect of the invention, pings are handled by shipping updated versions of resources directly between database servers without first being stored to disk, thus avoiding the I/O overhead associated with the disk intervention approach. Further, the difficulties associated with single-instance failure recovery are avoided by preventing a modified version of a resource from being replaced in cache until the modified resource or some successor thereof has been written to disk, even if the resource has been transferred to another cache.

[0021] For the purpose of explanation, a copy of a resource that cannot be replaced in cache is referred to herein as a "pinned" resource. The act of making a pinned resource replaceable is referred to as "releasing" the resource.

THE M AND W LOCK APPROACH

[0022] According to one aspect of the invention, the modify and write-to-disk permissions for a resource are separated. Thus, a database server that has permission to write an updated version of a resource from cache to disk does not necessarily have permission to update the resource. Conversely, a database server that has permission to modify a cached version of a resource does not necessarily have permission to write that cached version to disk.

[0023] According to one embodiment, this separation of permissions is enforced through the use of special locks. Specifically, the permission to modify a resource may be granted by a "M" lock, while the permission to write a resource to disk may be granted by a "W" lock. However, it should be noted that the use of M and W locks as described herein represents but one mechanism for preventing a transferred version of a resource from being replaced in cache until that version or a successor thereof is written to disk.

[0024] Referring to Figure 2, it illustrates the steps performed in response to a ping in a database system that uses M and W locks, according to one embodiment of the invention. At step 200, a database server that desires to modify a resource requests the M lock from the Master for the resource (i.e. the database server that manages the locks for the resource). At step 202, the Master instructs the database server currently holding the M lock for the resource (" the Holder")to transfer the M lock together with its cached version of the resource to the requesting database server via direct transfer over the communication channel(s) connecting the two servers (the "interconnect").

[0025] At step 204, the Holder sends the current version of the resource and the M lock to the Requestor. At step 206, the Holder informs the Master about the transfer of the M lock. At step 208, the Master updates the lock information for the resource to indicate that the Requestor now holds the M lock.

PI RESOURCES

[0026] The holder of the M lock does not necessarily have the W lock, and therefore may not have permission to write the version of the resource that is contained in its cache out to disk. The transferring database server (i.e. the database server that last held the M lock) therefore continues to pin its version of the resource in dynamic memory because it may be asked to write out its version to disk at some future point, as described below. The version of the resource that remains in the transferring database server will become out-of-date if the receiving database server modifies its copy of the resource. The transferring database server will not necessarily know when the receiving database server (or a successor thereof) modifies the resource, so from the time the transferring database server sends a copy of the resource, it treats its retained version as "potentially out-of-date". Such potentially out-of-date versions of a resource are referred to herein as past-image resources (PI resources).

RELEASING PI RESOURCES

[0027] After a cached version of a resource is released, it may be overwritten with new data. Typically, a dirty version of a resource may be released by writing the resource to disk. However, database servers with PI resources in cache do not necessarily have the right to store the PI resources to disk. One technique for releasing PI resources under these circumstances is illustrated in Figure 3.

[0028] Referring to Figure 3, when a database server wishes to release a PI resource in its cache, it sends a request for the W lock (step 300) to the distributed lock manager (DLM). In step 302, the DLM then orders the requesting database server, or some database server that has a later version of the resource (a successor) in its cache, to write the resource out to disk. The database server thus ordered to write the resource to disk is granted the W lock. After the database server that was granted the W lock writes the resource to disk, the database server releases the W lock.

[0029] The DLM then sends out a message to all database servers indicating the version of the resource written out (step 304), so that all earlier PI versions of the resource can be released (step 306). For example, assume that the version written to disk was modified at time T10. A database server with a version of the resource that was last modified at an earlier time T5 could now use the buffer in which it is stored for other data. A database server with a version that was modified at a later time T11, however, would have to continue to retain its version of the resource in its memory.

PING MANAGEMENT UNDER THE M AND W LOCK APPROACH

[0030] According to one embodiment of the invention, the M and W lock approach may be implemented to handle pings as shall now be described with reference to Figure 1. Referring to Figure 1, it is a block diagram that illustrates four database servers A, B, C and D, all of which have access to a database that contains a particular resource. At the time illustrated, database servers A, B and C all have versions of the resource. The version held in the cache of database server A is the most recently modified version of the resource (modified at time T10). The versions held in database servers B and C are PI versions of the resource. Database server D is the Master for the resource.

[0031] At this point, assume that another database server (the "Requestor") desires to modify the resource. The Requestor requests the modify lock from the Master. The Master sends a command to database server A to down-convert the lock (a "BAST") due to the conflicting request from the Requestor. In response to the down-convert command, the current image of the resource (whether clean or dirty) is shipped from database server A to the Requestor, together with a permission to modify the resource. The permission thus shipped does not include a permission to write the resource to disk.

[0032] When database server A passes the M lock to the Requestor, database server A downgrades his M lock to a "hold" lock (and "H lock"). The H lock indicates that the database server A is holding a pinned PI copy. Ownership of an H lock obligates the owner to keep the PI copy in its buffer cache, but does not give the database server any rights to write the PI copy to disk. There can be multiple concurrent H holders for the same resource, but not more than one database server at a time can write the resource, therefore only one database server can hold a W lock on the resource.

[0033] Prior to shipping the resource, database server A makes sure that the log is forced (i.e. that the recovery log generated for the changes made by database server A to the resource are durably stored). By passing the modification permission, database server A loses its own right to modify the resource. The copy of the resource (as it was just at the moment of shipping) is still kept at the shipping database server A. After the shipment of the resource, the copy of the resource retained in database server A is a PI resource.

COURTESY WRITES

[0034] After a database server ships a dirty resource directly to another database server, the retained copy of the resource becomes a pinned PI resource whose buffer cannot be used for another resource until released. The buffers that contain PI resources are referred to herein as PI buffers. These buffers occupy valuable space in the caches of the database servers, and eventually have to be reused for other data.

[0035] To replace PI buffers in the buffer cache (to be aged out or checkpointed) a new disk write protocol, referred to herein as "courtesy writes", is employed. According to the courtesy write protocol, when a database server needs to write a resource to disk, the database server sends the request to the DLM. The DLM selects a version of the resource to be written to disk, finds the database server that has the selected version, and causes that database server to write the resource to disk on behalf of the database server which initiated the write request. The database server that actually writes the resource to disk may be the database server which requested the write, or some other database server, depending on the latest trajectory of the resource.

[0036] Writing the selected version of the resource to disk releases all PI versions of the resource in all buffer caches of a cluster that are as old or older than the selected version that was written to disk. The criteria used to select the version that will be written to disk shall be described in greater detail hereafter. However, the selected version can be either the latest PI version known to the Master or the current version ("CURR") of the resource. One benefit of selecting a version other than the current version is that selection of another version leaves the current copy uninterruptedly available for modifications.

[0037] A database server that is holding a PI resource can write out its PI copy provided that it has acquired a W lock on the resource. The writes of the resource are decoupled from the migration of the CURR resource image among the various database servers.

EFFICIENCY FACTORS

[0038] There is no need to write a PI copy each time a resource is shipped to another database server. Therefore, the goal of durably storing resources is to keep the disk copies recent enough, and to keep the number of non-replaceable resources in the buffer caches reasonable. Various factors determine the efficiency of a system that employs the courtesy write protocol described above. Specifically, it is desirable to:

(1) minimize I/O activity caused by writing dirty resources to disk;
(2) keep the disk versions of resources current enough to speed up recovery operations after a failure; and
(3) prevent overflow of the buffer cache with pinned PI resources.

[0039] Maximizing the first criteria has a negative impact on the second and third criteria, and visa versa. Therefore, a trade off is necessary. According to one embodiment of the invention, a self-tuning algorithm may be used which combines different techniques of checkpointing (LRU mixed with occasional continuous

checkpointing) coupled with a control over the total IO budget.

THE NEWER-WRITE APPROACH

[0040] An alternative to the courtesy-write protocol described above is referred to herein as the write-newer approach. According to the write-newer approach, all database servers have permission to write their PI resources to disk. However, prior to doing so, a database server acquires a lock on the disk-based copy of the resource. After acquiring the lock, the database server compares the disk version with the PI version that it desires to write. If the disk version is older, then the PI version is written to disk. If the disk version is newer, then the PI version may be discarded and the buffer that it occupied may be reused.

[0041] Unlike the courtesy-write protocol, the newer-write approach allows a database server to release its own PI version, either by writing it to disk or determining that the disk version is newer. However, the newer-write approach increases contention for the lock of the disk-based copy, and may incur a disk-I/O that would not have been incurred with the courtesy-write approach.

PERMISSION STRINGS

[0042] Typical DLMs govern access to resources through the use of a limited number of lock modes, where the modes are either compatible or conflicting. According to one embodiment, the mechanism for governing access to resources is expanded to substitute lock modes with a collection of different kinds of permissions and obligations. The permissions and obligations may include, for example, the permission to write a resource, to modify a resource, to keep a resource in cache, etc. Specific permissions and obligations are described in greater detail below.

[0043] According to one embodiment, permissions and obligations are encoded in permission strings. A permission string might be augmented by a resource version number since many permissions are related to a version of a resource rather than to the resource itself. Two different permission strings are conflicting if they demand the same exclusive permission for the same version of the resource (e.g. current version for modification or a disk access for write). Otherwise they are compatible.

CONCURRENCY USING PERMISSION TRANSFERS

[0044] As mentioned above, when a resource is modified at one database server and is requested for further modifications by another database server, the Master instructs the database server that holds the current copy (CURR copy) of the resource to pass its M lock (the right to modify) together with the CURR copy of the resource to the other database server. Significantly, though the request for the M lock is sent to the master, the grant is done by some other database server (the previous M lock holder). This triangular messaging model deviates significantly from the traditional two-way communication where the response to a lock request is expected from the database server containing the lock manager to which the lock request was initially addressed.

[0045] According to one embodiment of the invention, when the holder of the CURR copy of a resource (e.g. database server A) passes the M lock to another database server, database server A notifies the Master that the M lock has been transferred. However, database server A does not wait for acknowledgment that the Master received the notification, but sends the CURR copy and the M lock prior to receiving such acknowledgement. By not waiting, the round trip communication between the master and database server A does not impose a delay on the transfer, thereby yielding a considerable saving on the protocol latencies.

[0046] Because permissions are transferred directly from the current holder of the permission to the requestor of the permission, the Master does not always know the exact global picture of the lock grants. Rather, the Master knows only about the trajectory of the M lock, about the database servers which just 'held it lately', but not about the exact location of the lock at any given time. According to one embodiment, this "lazy" notification scheme is applicable to the M locks but not to W, X, or S locks (or their counterparts). Various embodiments of a locking scheme are described in greater detail below.

FAILURE RECOVERY

[0047] Within the context of the present invention, a database server is said to have failed if a cache associated with the server becomes inaccessible. Database systems that employ the direct, inter-server shipment of dirty resources using the techniques described herein avoid the need for merging recovery logs in response to a single-server failure. According to one embodiment, single-server failures are handled as illustrated in Figure 4. Referring to Figure 4, upon a single-database server failure, the recovery process performs the following for each resource held in the cache of the failed database server:

(step 400) determine the database server that held the latest version of the resource;
(step 402) if the database server determined in step 400 is not the failed database server, then (step 404) the determined database server writes its cached version of the resource to disk and (step 406) all PI versions of the resource are released. This version will have all the committed changes made to the resource (including those made by the failed database server) and thus no recovery log of any database server need be applied.

[0048] If the database server determined in step 402 is the failed database server, then (step 408) the database server holding the latest PI version of the resource writes out its cached version of the resource to disk and (step 410) all previous PI versions are released. The version written out to disk will have the committed changes made to the resource by all database servers except the failed database server. The recovery log of the failed database server is applied (step 412) to recover the committed changes made by the failed database server.

[0049] Alternatively, the latest PI version of the resource may be used as the starting point for recovering the current version in cache, rather than on disk. Specifically, the appropriate records from the recovery log of the failed database server may be applied directly to the latest PI version that resides in cache, thus reconstructing the current version in the cache of the database server that holds the latest PI version.

MULTIPLE DATABASE SERVER FAILURE

[0050] In case of a multiple server failure, when neither the latest PI copy nor any CURR copy have survived, it may happen that the changes made to the resource are spread over multiple logs of the failed database servers. Under these conditions, the logs of the failed database servers must be merged. However, only the logs of the failed database servers must be merged, and not logs of all database servers. Thus, the amount of work required for recovery is proportional to the extent of the failure and not to the size of the total configuration.

[0051] In systems where it is possible to determine which failed database servers updated the resource, only the logs of the failed database servers that updated the resource need to be merged and applied. Similarly, in systems where it is possible to determine which failed database servers updated the resource subsequent to the durably stored version of the resource, only the logs of the failed database servers that updated the resource subsequent to the durably stored version of the resource need to be merged and applied.

EXEMPLARY OPERATION

[0052] For the purpose of explanation, an exemplary series of resource transfers shall be described with reference to Figure 1. During the series of transfers, a resource is accessed at multiple database servers. Specifically, the resource is shipped along a cluster nodes for modifications, and then a checkpoint at one of the database servers causes a physical I/O of this resource.

[0053] Referring again to Figure 1, there are 4 database servers: A,B,C, and D. Database server D is the master of the resource. Database server C first modifies the resource. Database server C has resource version 8. At this point, database server C also has an M lock (an exclusive modification right) on this resource.

[0054] Assume that at this point, database server B wants to modify the resource that database server C currently holds. Database server B sends a request (1) for an M lock on the resource. Database server D puts the request on a modifiers queue associated with the resource and instructs (message 2: BAST) database server C to:

(a) pass modification permission (M lock) to database server B,
(b) send current image of the resource to database server B, and
(c) downgrade database server C's M lock to an H lock.

[0055] After this downgrade operation, C is obligated to keep its version of the resource (the PI copy) in its buffer cache.

[0056] Database server C performs the requested operations, and may additionally force the log on the new changes. In addition, database server C lazily notifies (3 AckM) the Master that it has performed the operations (AST). The notification also informs the Master that database server C keeps version 8. Database server C does not wait for any acknowledgment from the Master. Consequently, it is possible that database server B gets an M lock before the Master knows about it.

[0057] Meanwhile, assume that database server A also decides to modify the resource. Database server A sends a message (4) to database server D. This message may arrive before the asynchronous notification from database server C to database server D.

[0058] Database server D (the Master) sends a message (5) to database server B, the last known modifier of this resource, to pass the resource (after B gets and modifies it) to database server A. Note that database server D does not know whether the resource is there or not yet. But database server D knows that the resource will eventually arrive at B.

[0059] After database server B gets the resource and makes the intended changes (now B has version 9 of the resource), it downgrades its own lock to H, sends (6) the current version of the resource ("CURR resource") to database server A together with the M lock. Database server B also sends a lazy notification (6 AckM) to the Master.

[0060] While this resource is being modified at database server A, assume that a checkpointing mechanism at database server C decides to write the resource to disk. Regarding the asynchronous events described above, assume that both 3AckM and 6 AckM have already arrived to the master. The operations performed in response to the checkpointing operation are illustrated with reference to Figure 5.

[0061] Referring to Figure 5, since database server C holds an H lock on version 8, which does not include a writing privilege, database server C sends message 1 to the Master (D) requesting the W (write) lock for its

version. At this point in time, the Master knows that the resource was shipped to database server A (assuming that the acknowledgments have arrived). Database server D sends an (unsolicited) W lock to database server A (2 BastW) with the instruction to write the resource.

[0062] In the general case, this instruction is sent to the last database server whose send notification has arrived (or to the database server which is supposed to receive the resource from the last known sender). Database server A writes (3) its version of the resource. The resource written by database server A is version 10 of the resource. By this time, the current copy of the resource might be somewhere else if additional requestors demanded the resource. The disk acknowledges when the write is completed (4Ack).

[0063] When the write completes, database server A provides database server D with the information that version 10 is now on disk (5 AckW). Database server A voluntarily downgrades its W lock (which it did not ask for in the first place).

[0064] The Master (D) goes to database server C and, instead of granting the requested W lock, notifies C that the write completed (6). The Master communicates the current disk version number to the holders of all PI copies, so that all earlier PI copies at C can be released. In this scenario, since database server C has no PI copies older than 10, it downconverts database server C's lock to NULL.

[0065] The Master also sends an acknowledgment message to database server B instructing database server B to release its PI copies which are earlier than 10 (7AckW(10)).

THE DISTRIBUTED LOCK MANAGER

[0066] In contrast with conventional DLM logic, the Master in a system that implements the direct-shipping techniques described herein may have incomplete information about lock states at the database servers. According to one embodiment, the Master of a resource maintains the following information and data structures:

(1) a queue of CURR copy requestors (either for modification or for shared access) (the upper limit on the queue length is the number of database servers in the cluster). This queue is referred to herein as the Current Request Queue (CQ).

(2) when a resource is sent to another CURR requestor, the senders lazily (asynchronously in a sense that they do not wait for a acknowledgment) notify the Master about the event. Master keeps track of the last few senders. This is a pointer on the CQ.

(3) the version number of the latest resource version on disk.

(4) W lock grants and a W requests queue.

According to one embodiment, W permission is synchronous: it is granted only by the master, and the master ensures that there is not more than one writer in the cluster for this resource. The Master can make the next grant only after being notified that the previous write completed and the W lock was released. If there are more than one modifier, a W lock is given for the duration of the write and voluntarily released after the write. If there is only one modifier, the modifier can keep the W permission.

(5) a list of H lock holders with their respective resource version numbers. This provides information (though possibly incomplete) about the PI copies in buffer caches.

DISK WARM UP

[0067] Since the direct-shipment techniques described herein significantly segregate the life cycles of the buffer cache images of the resources and the disk images, there is a need to bridge this gap on recovery. According to one embodiment, a new step of recovery, between DLM recovery and buffer cache recovery, is added. This new recovery step is referred to herein as 'disk warm up'.

[0068] Although during normal cache operations a master of a resource has only approximate knowledge of the resource location and about the availability of PI and CURR copies, on DLM recovery (which precedes cache recovery), the master of a resource collects complete information about the availability of the latest PI and CURR copies in the buffer caches of surviving database servers. This is true whether or not the master of the resource is a new master (if before the failure the resource was mastered on a failed database server) or a surviving master.

[0069] After collecting this information, the Master knows which database server possesses the latest copy of the resource. At '*disk warm up*' stage, the master issues a W lock to the owner of this latest copy of the resource (CURR if it is available, and latest PI copy if the CURR copy disappeared together with the failed database server). The master then instructs this database server to write the resource to disk. When the write completes, all other database servers convert their H locks to NULL locks (because the written copy is the latest available). After those locks have been converted, cache recovery can proceed as normal.

[0070] Some optimizations are possible during the disk warm up stage. For example, the resource does not necessarily have to be written to disk if the latest image is in the buffer cache of the database server performing recovery.

ALTERNATIVES TO LOCK-BASED SCHEME

[0071] Various techniques for directly shipping dirty copies of resources between database servers have been described in the context of a locking scheme that

uses special types of locks (M, W and H locks). Specifically, these special locks are used to ensure that (1) only the server with the current version of the resource modifies the resource, (2) all servers keep their PI versions of the resource until the same version or a newer version of the resource is written to disk, and (3) the disk-based version of the resource is not overwritten by an older version of the resource.

[0072] However, a lock-based access control scheme is merely one context in which the present invention may be implemented. For example, those same three rules may be enforced using any variety of access control schemes. Thus, present invention is not limited to any particular type of access control scheme.

[0073] For example, rather than governing access to a resource based on locks, access may be governed by tokens, where each token represents a particular type of permission. The tokens for a particular resource may be transferred among the parallel servers in a way that ensures that the three rules stated above are enforced.

[0074] Similarly, the rules may be enforced using a state-based scheme. In a state-based scheme, a version of a resource changes state in response to events, where the state of a version dictates the type of actions that may be performed on the version. For example, a database server receives the current version of a resource in its "current" state. The current state allows modification of the resource, and writing to disk of the resource. When a database server transfers the current version of the resource to another node, the retained version changes to a "PI writeable" state. In the PI writeable state, the version (1) cannot be modified, (2) cannot be overwritten, but (3) can be written to disk. When any version of the resource is written to disk, all versions that are in PI writeable state that are the same or older than the version that was written to disk are placed in a "PI released" state. In the PI released state, versions can be overwritten, but cannot be modified or written to disk.

HARDWARE OVERVIEW

[0075] Figure 6 is a block diagram that illustrates a computer system 600 upon which an embodiment of the invention may be implemented. Computer system 600 includes a bus 602 or other communication mechanism for communicating information, and a processor 604 coupled with bus 602 for processing information. Computer system 600 also includes a main memory 606, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 602 for storing information and instructions to be executed by processor 604. Main memory 606 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 604. Computer system 600 further includes a read only memory (ROM) 608 or other static storage device coupled to bus 602 for storing static information and instructions for processor 604. A storage device 610, such as a magnetic disk or optical disk, is provided and coupled to bus 602 for storing information and instructions.

[0076] Computer system 600 may be coupled via bus 602 to a display 612, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 614, including alphanumeric and other keys, is coupled to bus 602 for communicating information and command selections to processor 604. Another type of user input device is cursor control 616, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 604 and for controlling cursor movement on display 612. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0077] The invention is related to the use of computer system 600 for reducing the overhead associated with a ping. According to one embodiment of the invention, the overhead associated with a ping is reduced by computer system 600 in response to processor 604 executing one or more sequences of one or more instructions contained in main memory 606. Such instructions may be read into main memory 606 from another computer-readable medium, such as storage device 610. Execution of the sequences of instructions contained in main memory 606 causes processor 604 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0078] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 604 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 610. Volatile media includes dynamic memory, such as main memory 606. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 602. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0079] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0080] Various forms of computer readable media may be involved in carrying one or more sequences of

one or more instructions to processor 604 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 600 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 602. Bus 602 carries the data to main memory 606, from which processor 604 retrieves and executes the instructions. The instructions received by main memory 606 may optionally be stored on storage device 610 either before or after execution by processor 604.

[0081] Computer system 600 belongs to a shared disk system in which data on one or more storage devices (e.g. disk drives 655) are accessible to both computer system 600 and to one or more other CPUs (e.g. CPU 651). In the illustrated system, shared access to the disk drives 655 is provided by a system area network 653. However, various mechanisms may alternatively be used to provide shared access.

[0082] Computer system 600 also includes a communication interface 618 coupled to bus 602. Communication interface 618 provides a two-way data communication coupling to a network link 620 that is connected to a local network 622. For example, communication interface 618 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 618 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 618 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0083] Network link 620 typically provides data communication through one or more networks to other data devices. For example, network link 620 may provide a connection through local network 622 to a host computer 624 or to data equipment operated by an Internet Service Provider (ISP) 626. ISP 626 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 628. Local network 622 and Internet 628 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 620 and through communication interface 618, which carry the digital data to and from computer system 600, are exemplary forms of carrier waves transporting the information.

[0084] Computer system 600 can send messages and receive data, including program code, through the network(s), network link 620 and communication interface 618. In the Internet example, a server 630 might transmit a requested code for an application program through Internet 628, ISP 626, local network 622 and communication interface 618.

[0085] The received code may be executed by processor 604 as it is received, and/or stored in storage device 610, or other non-volatile storage for later execution. In this manner, computer system 600 may obtain application code in the form of a carrier wave.

[0086] While techniques for handling pings have been described herein with reference to pings that occur when multiple database servers have access to a common persistent storage device, these techniques are not restricted to this context. Specifically, these techniques may be applied in any environment where a process associated with one cache may require a resource whose current version is located in another cache. Such environments include, for example, environments in which text servers on different nodes have access to the same text material, environments in which media servers on different nodes have access to the same video data, etc.

[0087] Handling pings using the techniques described herein provides efficient inter-database server transfer of resources so uptime performance scales well with increasing number of database servers, and users per database server. In addition, the techniques result in efficient recovery from single-database server failures (the most common type of failure) that scales well with increasing number of database servers.

[0088] Significantly, the techniques described herein handle pings by sending resources via the IPC transport, not through disk intervention. Consequently, disk I/Os for resources that result in a ping are substantially eliminated. A synchronous I/O is involved only as long as it is needed for the log force. In addition, while disk I/O is incurred for checkpointing and buffer cache replacement, such I/O does not slow down the buffer shipment across the cluster.

[0089] The direct shipping techniques described herein also tend to reduced the number of context switches incurred by a ping. Specifically, the sequence of round trip messages between the participants of the protocol (requestor and holder) and the Master, is substituted by the communication triangle: Requestor, Master, Holder, Requestor.

[0090] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

Claims

1. A method for transferring a resource from a first cache to a second cache, the method comprising

the steps of:

retaining a first copy of the resource in said first cache while transferring a second copy of the resource from the first cache to the second cache without first durably storing said resource from said first cache to a persistent storage (655); and
retaining at least one copy of the resource in said first cache until said first copy of the resource or a successor thereof is durably stored.

2. The method of Claim 1 wherein said first cache is a cache maintained by a first database server (A, B, C, D) and said second cache is a cache maintained by a second (A, B, C, D) database server.

3. The method of Claim 1 further comprising the steps of:

allowing said first copy of said resource to be modified in said first cache prior to transferring said second copy to said second cache; and
preventing said first copy of said resource from being modified after transferring said second copy to said second cache.

4. The method of Claim 1 further comprising the steps of:

after transferring said second copy to said second cache, sending a request for permission to release said first copy;
in response to said request, causing said first copy or a successor thereof to be durably stored (302); and
in response to said successor being durably stored, sending a message that indicates that said first copy can be released (304).

5. The method of Claim 4 wherein:

the step of sending a request for permission to release said first copy is performed by a sending process; and
the step of causing said first copy or a successor thereof to be durably stored includes the step of causing a process other than the sending process to store a successor to said first copy of said resource (302).

6. The method of Claim 1 wherein the step of retaining at least one copy of the resource in said first cache includes the steps of:

prior to attempting to durably store said first copy, determining whether a durably stored copy of said resource is more recent than said first copy;
if said durably stored copy is more recent than said first copy, then releasing said first copy without durably storing said first copy; and
if said durably stored copy is not more recent than said first copy, then durably storing said first copy.

7. The method of claim 3 further comprising the step of transferring a modify permission from a sending process associated with the first cache (204) to a receiving process associated with the second cache along with said second copy of said resource.

8. The method of Claim 7 wherein:

permissions for accessing said resource are governed by a master (D); and
the step of transferring said modify permission to the receiving process is performed prior to receiving acknowledgement from said master (D) for transfer of said modify permission to said receiving process (204).

9. The method of Claim 1 further comprising the steps of:

transferring a modify permission from a sending process associated with said first cache to a receiving process (204) associated with said second cache along with said second copy of the resource;
wherein permissions for accessing said resource are governed by a master (D); and
wherein the step of transferring said modify permission to said receiving process is performed prior to said master (D) receiving acknowledgement of the transfer of said modify permission to said receiving process (204).

10. The method of Claim 1 further comprising the steps of:

a receiving process associated with said second cache sending a request for said resource to a master (D) of said resource;
in response to said request from said receiving process, said master (D) of said resource sending a message to a sending process associated with said first cache; and
said sending process transferring said second copy to said receiving process in response to said message from said master (D).

11. The method of Claim 1 further comprising performing the following steps after the step of transferring said second copy to said second cache:

a sending process associated with said first cache requesting a lock (300) from a lock manager, wherein said lock grants permission to write said resource to disk but not permission to modify said resource;
said lock manager selecting a process that has a version of said resource that is at least as recent as said first copy;
said lock manager granting said lock to said selected process; and
said selected process writing said version of said resource to disk.

12. The method of Claim 11 further comprising the step of, in response to said version of said resource being written to disk (302), said lock manager causes all versions of said resource that are older than said version to be released (306).

13. The method of Claim 1 further comprising the steps of, after a failure of a cache that holds a dirty copy of said resource:

determining whether the failed cache held the latest version of the resource (402);
if the failed cache held the latest version of the resource, then

writing a latest past image of the resource to disk (408);
releasing all previous past images of the resource (410); and
applying a recovery log of said failed cache to reconstruct the latest version of the resource (412).

14. The method of Claim 13 further comprising the steps of:

if the failed cache did not hold the latest version of the resource, then

writing the latest version of the resource to disk (404); and
releasing all past images of the resource (406).

15. The method of Claim 1 further comprising the steps of, after a failure of a plurality of caches that hold dirty versions of said resource:

determining whether any of the failed caches held the latest version of the resource (400); and
if any of the failed caches held the latest version of the resource, then

merging and applying the recovery logs of said failed caches to reconstruct the latest version of the resource.

16. The method of Claim 1, wherein the second copy of the resource is a dirty version of the resource, wherein the first cache and the second cache belong to a plurality of caches, and further comprising the steps of:

maintaining separate recovery logs for each cache of said plurality of caches; and
when a cache of said plurality of caches fails, recovering the failed cache based on the recovery log associated with said failed cache without inspecting the separate recovery logs of the other caches of said plurality of caches.

17. The method of Claim 16 wherein each cache of said plurality of caches is a cache maintained by a separate database server (A, B, C, D) of a plurality of database (A, B, C, D) servers.

18. The method of Claim 16 wherein:

the method further comprising the steps of:

allowing said first copy of said resource to be modified in said first cache prior to transferring said second copy to said second cache; and
preventing said first copy of said resource from being modified after transferring said second copy to said second cache.

19. The method of Claim 1, wherein the first cache resides on a first node of a plurality of nodes (A, B, C, D), and the second cache resides on a second node of said plurality of nodes (A, B, C, D), and further comprising the steps of:

modifying said resource in the first cache of the first node of a plurality of nodes (A, B, C, D) to create a modified version of said resource;
maintaining a checkpoint for said first node that indicates where to begin work when said first node fails;
retaining a first copy of said modified version in said first cache while transferring a second copy of said modified version from the first cache to the second cache of the second node of the plurality of nodes (A, B, C, D) without first durably storing said modified version from said first cache to a persistent storage (655); and
in response to an indication that another node of said plurality of nodes (A, B, C, D) durably stored a version of said resource that is at least as recent as said modified version, advancing the checkpoint.

20. The method of Claim 19, further comprising the step of:

retaining at least one copy of said modified version in said first cache until said first copy of said modified version or a successor thereof is durably stored.

21. The method of Claim 19, further comprising the step of:

in response to the indication that another node of said plurality of nodes durably stored the version of said resource that is at least as recent as said modified version, releasing said resource at the first node (306).

22. A computer-readable medium carrying one or more sequences of instructions for transferring a resource from a first cache to a second cache, wherein execution of the one or more sequences of instructions by one or more processors causes the one or more processors to perform the steps of the method recited in any one of Claims 1-21.

23. A system for transferring a resource, the system comprising:

a first node that has a first cache that is communicatively coupled to a second cache from among one or more other caches that are included in one or more other nodes;
wherein said first node is configured to retain a first copy of the resource in the first cache while transferring a second copy of the resource from the first cache to the second cache without first durably storing said resource from said first cache to a persistent storage (655); and
wherein said first node is configured to retain at least one copy of the resource in

said first cache until said first copy of the resource or a successor thereof is durably stored.

24. The system of Claim 23 wherein said first node is a first database server (A, B, C, D) and at least one of said one or more other nodes is second database server (A, B, C, D) that includes said second cache.

25. The system of Claim 23, wherein

said first node is configured to allow said first copy of said resource to be modified in said first cache prior to transferring said second copy to said second cache; and
said first node is configured to prevent said first copy of said resource from being modified after transferring said second copy to said second cache.

26. The system of Claim 23, wherein:

said first node is configured to send a request to a master node (D) for permission to release said first copy, after transferring said second copy to said second cache; and
said first node is configured to receive a message from said master node (D) that indicates that said first copy can be released after said master node (D) causes, in response to said request, said first copy or a successor thereof to be durably stored.

27. The system of Claim 26, wherein

said first node comprises a sending process that is configured to send the request for permission to release said first copy; and
a process other than the sending process stores a successor to said first copy of said resource.

28. The system of Claim 23 wherein said first node is configured to retain at least one copy of the resource in said first cache by:

prior to attempting to durably store said first copy, determining whether a durably stored copy of said resource is more recent than said first copy;
if said durably stored copy is more recent than said first copy, then releasing said first copy without durably storing said first copy; and
if said durably stored copy is not more recent than said first copy, then durably storing said first copy.

29. The system of Claim 25, wherein said first node is configured to transfer a modify permission from a sending process associated with the first cache to a receiving process associated with the second cache along with said second copy of said resource.

30. The system of Claim 29 wherein:

permissions for accessing said resource are governed by a master node (D); and
said first node is configured to transfer said modify permission to the receiving process prior to said first node receiving acknowledgement from said master node (D) for transfer of said modify permission to said receiving process.

31. The system of Claim 23 wherein:

said first node is configured to transfer a modify permission from a sending process associated with said first cache to a receiving process associated with said second cache along with said second copy of said resource;
permissions for accessing said resource are governed by a master (D); and
the transfer of said modify permission to said receiving process is performed prior to said master (D) receiving acknowledgement of the transfer of said modify permission to said receiving process.

32. The system of Claim 23, wherein:

said first node includes a sending process associated with said first cache, wherein said sending process is configured to receive a message from a master node (D) that receives a request for said resource from a receiving process associated with said second cache; and
said sending process transfers said second copy to said receiving process in response to said message from said master node (D).

33. The system of Claim 23, wherein:

said first node includes a sending process that is configured to request a lock from a lock manager after said second copy is transferred to said second cache, wherein said lock grants permission to write said resource to disk (655) but not permission to modify said resource; and
said lock manager selects a process that has a version of said resource that is at least as recent as said first copy and grants said lock to said selected process to cause said selected process to write said version of said resource to disk (655).

34. The system of Claim 23 wherein said lock manager causes all versions of said resource that are older than said version to be released, in response to said version of said resource being written to disk (655).

35. The system of Claim 23 further comprising:

a master node (D) that is configured to determine, after a failure of a cache that holds a dirty copy of said resource, whether a failed cache held the latest version of the resource; and
wherein, if the failed cache held the latest version of the resource, the master node (D) is configured to cause a latest past image of the resource to be written to disk (655), to cause all previous past images of the resource to be released, and to cause a recovery log of said failed cache to be applied to reconstruct the latest version of the resource.

36. The system of Claim 35 wherein the master node (D) is configured to, if the failed cache did not hold the latest version of the resource, cause the latest version of the resource to be written to disk (655), and to cause all past images of the resource to be released.

37. The system of Claim 23 further comprising:

a master node (D) that is configured to, after a failure of a plurality of caches that hold dirty versions of said resource, determine whether any of the failed caches held the latest version of the resource, and if any of the failed caches held the latest version of the resource, then merge and apply the recovery logs of said failed caches to reconstruct the latest version of the resource.

38. The system of Claim 23,
wherein the first cache and the second cache belong to a plurality of caches, wherein a separate recovery log is maintained for each cache of said plurality of caches, and said system further comprises:

a master node (D) that is configured to recover a failed cache from among said plurality of caches based on the recovery log associated with said failed cache without inspecting the separate recovery logs of the other caches of said plurality of caches.

39. The system of Claim 38, wherein each cache of said plurality of caches is a cache maintained by a separate database server of a plurality of database servers (A, B, C, D).

40. The system of Claim 38, wherein:

the first node is configured to allow said first copy of said resource to be modified in said first cache prior to transferring said second copy to said second cache; and
the first node is configured to prevent said first copy of said resource from being modified after transferring said second copy to said second cache.

41. The system of Claim 23,
wherein said first node is configured to:

modify said resource in the first cache of said first node to create a modified version of said resource;
maintain a checkpoint for said first node that indicates where to begin work when said first

node fails;
retain a first copy of said modified version in said first cache while transferring a second copy of said modified version from the first cache to the second cache without first durably storing said modified version from said first cache to a persistent storage; and
in response to an indication that another node of said plurality of nodes (A, B, C, D) durably stored a version of said resource that is at least as recent as said modified version, advance the checkpoint.

42. The system of Claim 41, wherein the first node is further configured to:

retain at least one copy of said modified version in said first cache until said first copy of said modified version or a successor thereof is durably stored.

43. The system of Claim 41, wherein the first node is further configured to:

in response to the indication that another node of said plurality of nodes (A, B, C, D) durably stored the version of said resource that is at least as recent as said modified version, release said resource at the first node.

44. The system of Claim 23,
wherein said first node is configured to prevent said first copy from being replaced in said first cache until said first copy of the resource or a successor thereof is durably stored.

Patentansprüche

1. Verfahren zum Übertragen einer Ressource von einem ersten. Zwischenspeicher zu einem zweiten Zwischenspeicher,
wobei das Verfahren die Schritte aufweist:

Beibehalten einer ersten Kopie der Ressource in dem ersten Zwischenspeicher während des Übertragens einer zweiten Kopie der Ressource von dem ersten Zwischenspeicher zu dem zweiten Zwischenspeicher, ohne die Ressource von dem ersten Zwischenspeicher vorher dauerhaft in einen persistenten Speicher (655) zu speichern; und
Beibehalten mindestens einer Kopie der Ressource in dem ersten Zwischenspeicher bis die erste Kopie der Ressource oder ein Versionsnachfolger von ihr dauerhaft gespeichert ist.

2. Verfahren gemäß Anspruch 1, wobei der erste Zwischenspeicher ein Zwischenspeicher ist, der von einem ersten Datenbankserver (A,B,C,D) verwaltet wird, und der zweite Zwischenspeicher ein Zwischenspeicher ist, der von einem zweiten Datenbankserver (A,B,C,D) verwaltet wird.

3. Verfahren gemäß Anspruch 1, ferner aufweisend die Schritte:

Ermöglichen, dass die erste Kopie der Ressource in dem ersten Zwischenspeicher verändert wird, bevor die zweite Kopie zu dem zweiten Zwischenspeicher übertragen wird; und
Verhindern, dass die erste Kopie der Ressource verändert wird, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen worden ist.

4. Verfahren gemäß Anspruch 1, ferner aufweisend die Schritte:

Senden einer Anforderung für die Genehmigung zum Freigeben der ersten Kopie, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen worden ist;
Veranlassen, dass die erste Kopie oder ein Versionsnachfolger von ihr als Reaktion auf die Anforderung dauerhaft gespeichert wird (302); und
Senden einer Nachricht, die anzeigt, dass die erste Kopie freigegeben werden kann (304), als Reaktion darauf, dass der Versionsnachfolger dauerhaft gespeichert ist.

5. Verfahren gemäß Anspruch 4, wobei:

der Schritt des Sendens einer Anforderung für die Genehmigung zum Freigeben der ersten Kopie mittels eines Sendeprozesses durchgeführt wird; und
der Schritt des Veranlassens, dass die erste Kopie oder ein Versionsnachfolger von ihr dauerhaft gespeichert wird, den Schritt des Veranlassens, dass ein Prozess, der ein anderer als der Sendeprozess ist, einen Versionsnachfolger der ersten Kopie der Ressource speichert, aufweist (302).

6. Verfahren gemäß Anspruch 1, wobei der Schritt des Beibehaltens mindestens einer Kopie der Ressource in dem ersten Zwischenspeicher die Schritte aufweist:

Bestimmen, ob eine dauerhaft gespeicherte Kopie der Ressource neuer als die erste Kopie ist, bevor versucht wird die erste Kopie dauerhaft zu speichern;
Freigeben der ersten Kopie ohne dauerhaftes

Speichern der ersten Kopie, falls die dauerhaft gespeicherte Kopie neuer als die erste Kopie ist; und
dauerhaftes Speichern der ersten Kopie, falls die dauerhaft gespeicherte Kopie nicht neuer als die erste Kopie ist.

7. Verfahren gemäß Anspruch 3, ferner aufweisend den Schritt des Übertragens einer Änderungsgenehmigung von einem Sendeprozess, der dem ersten Zwischenspeicher zugeordnet ist (204), zu einem Empfangsprozess, der dem zweiten Zwischenspeicher zugeordnet ist, zusammen mit der zweiten Kopie der Ressource.

8. Verfahren gemäß Anspruch 7, wobei:

Genehmigungen für das Zugreifen auf die Ressource mittels eines Hauptservers (D) verwaltet werden; und
der Schritt des Übertragens der Änderungsgenehmigung zu dem Empfangsprozess vor dem Empfangen der Bestätigung von dem Hauptserver (D) für die Übertragung der Änderungsgenehmigung zu dem Empfangsprozess durchgeführt wird (204).

9. Verfahren gemäß Anspruch 1, ferner aufweisend die Schritte:

Übertragen einer Änderungsgenehmigung von einem Sendeprozess, der dem ersten Zwischenspeicher zugeordnet ist, an einen Empfangsprozess (204), der dem zweiten Zwischenspeicher zugeordnet ist, zusammen mit der zweiten Kopie der Ressource;
wobei Genehmigungen für das Zugreifen auf die Ressource mittels eines Hauptservers (D) verwaltet werden; und
wobei der Schritt des Übertragens der Änderungsgenehmigung zu dem Empfangsprozess durchgeführt wird, bevor der Hauptserver (D) die Bestätigung für die Übertragung der Änderungsgenehmigung zu dem Empfangsprozess empfängt (204).

10. Verfahren gemäß Anspruch 1, ferner aufweisend die Schritte:

ein Empfangsprozess, der dem zweiten Zwischenspeicher zugeordnet ist, sendet eine Anforderung für die Ressource an einen Hauptserver (D) der Ressource;
der Hauptserver (D) der Ressource sendet als Reaktion auf die Anforderung des Empfangsprozesses eine Nachricht zu einem Sendeprozess, der dem ersten Zwischenspeicher zugeordnet ist; und
der Sendeprozess überträgt die zweite Kopie zu dem Empfangsprozess als Reaktion auf die Nachricht von dem Hauptserver (D).

11. Verfahren gemäß Anspruch 1, das ferner nach dem Schritt der Übertragung der zweiten Kopie zu dem zweiten Zwischenspeicher die Ausführung der folgenden Schritte aufweist:

ein Sendeprozess, der dem ersten Zwischenspeicher zugeordnet ist, fordert eine Sperre (300) von einem Sperrmanager an, wobei die Sperre die Genehmigung zum Schreiben der Ressource auf Platte erteilt, aber nicht die Genehmigung zur Änderung der Ressource;
der Sperrmanager wählt einen Prozess aus, der eine Version der Ressource aufweist, die mindestens so neu wie die erste Kopie ist;
der Sperrmanager erteilt dem ausgewählten Prozess eine Sperre; und
der ausgewählte Prozess schreibt die Version der Ressource auf Platte.

12. Verfahren gemäß Anspruch 11, ferner aufweisend den Schritt, dass der Sperrmanager als Reaktion auf das Schreiben der Version der Ressource auf Platte (302), veranlasst, dass alle Versionen der Ressource, die älter als die Version sind, freigegeben werden (306).

13. Verfahren gemäß Anspruch 1, das ferner, nach einem Ausfall eines Zwischenspeichers, der eine verschmutzte Kopie der Ressource enthält, die Schritte aufweist:

Bestimmen, ob der ausgefallene Zwischenspeicher die letzte Version der Ressource enthalten hat (402);
falls der ausgefallene Zwischenspeicher die letzte Version der Ressource enthalten hat, dann:

Schreiben der letzten früheren Abbild-Kopie der Ressource auf Platte (408);
Freigeben aller vorherigen früheren Abbild-Kopien der Ressource (410); und
Verwenden eines Wiederherstellungsprotokolls des ausgefallenen Zwischenspeichers, um die letzte Version der Ressource zu rekonstruieren (412).

14. Verfahren gemäß Anspruch 13, ferner aufweisend die Schritte:

falls der ausgefallene Zwischenspeicher nicht die letzte Version der Ressource enthalten hat, dann

Schreiben der letzten Version der Ressource auf die Platte (404); und
Freigeben aller früheren Abbild-Kopien der Ressource (406).

15. Verfahren gemäß Anspruch 1, das ferner, nach einem Ausfall einer Mehrzahl von Zwischenspeichern, die verschmutzte Versionen der Ressource enthalten, die Schritte aufweist:

Bestimmen, ob einer der ausgefallenen Zwischenspeicher die letzte Version der Ressource enthalten hat (400); und
falls irgendeiner der ausgefallenen Zwischenspeicher die letzte Version der Ressource enthalten hat, dann:

Zusammenführen und Verwenden der Wiederherstellungsprotokolle der ausgefallenen Zwischenspeicher, um die letzte Version der Ressource zu rekonstruieren.

16. Verfahren gemäß Anspruch 1, wobei die zweite Kopie der Ressource eine verschmutzte Version der Ressource ist, wobei der erste Zwischenspeicher und der zweite Zwischenspeicher zu einer Mehrzahl von Zwischenspeichern gehören, das ferner die Schritte aufweist:

Verwalten von separaten Wiederherstellungsprotokollen für jeden Zwischenspeicher der Mehrzahl von Zwischenspeichern; und
wenn ein Zwischenspeicher der Mehrzahl von Zwischenspeichern ausgefällt, Wiederherstellen des ausgefallenen Zwischenspeichers basierend auf dem Wiederherstellungsprotokoll, das dem ausgefallenen Zwischenspeicher zugeordnet ist, ohne die separaten Wiederherstellungsprotokolle der anderen Zwischenspeicher der Mehrzahl von Zwischenspeichern zu überprüfen.

17. Verfahren gemäß Anspruch 16, wobei jeder Zwischenspeicher der Mehrzahl von Zwischenspeichern ein Zwischenspeicher ist, der von einem separaten Datenbank-Server (A, B, C, D) einer Mehrzahl von Datenbank-Servern (A, B, C, D) verwaltet wird.

18. Verfahren gemäß Anspruch 16, wobei das Verfahren ferner die Schritte aufweist:

Ermöglichen, dass die erste Kopie der Ressource in dem ersten Zwischenspeicher verändert wird, bevor die zweite Kopie zu dem zweiten Zwischenspeicher übertragen wird; und
Verhindern, dass die erste Kopie der Ressource verändert wird, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen worden ist.

19. Verfahren gemäß Anspruch 1, wobei sich der erste Zwischenspeicher auf einem ersten Knoten einer Mehrzahl von Knoten (A, B, C, D) befindet und der zweite Zwischenspeicher sich auf einem zweiten Knoten der Mehrzahl von Knoten (A, B, C, D) befindet, und das ferner die Schritte aufweist:

Verändern der Ressource in dem ersten Zwischenspeicher des ersten Knotens einer Mehrzahl von Knoten (A, B, C, D), um eine geänderte Version der Ressource zu erzeugen;
Verwalten eines Kontrollpunkts für den ersten Knoten, der anzeigt, wo mit der Arbeit begonnen werden muss, wenn der erste Knoten ausfällt;
Beibehalten einer ersten Kopie der geänderten Version in dem ersten Zwischenspeicher während des Übertragens einer zweiten Kopie der geänderten Version von dem ersten Zwischenspeicher zu dem zweiten Zwischenspeicher des zweiten Knotens der Mehrzahl von Knoten (A, B, C, D), ohne die geänderte Version von dem ersten Zwischenspeicher vorher dauerhaft in einen persistenten Speicher (655) zu speichern; und
Vorversetzen des Kontrollpunkts als Reaktion auf den Hinweis, das ein anderer Knoten der Mehrzahl von Knoten (A, B, C, D) eine Version der Ressource dauerhaft gespeichert hat, die mindestens so neu wie die geänderte Version ist.

20. Verfahren gemäß Anspruch 19, ferner aufweisend die Schritte:

Beibehalten mindestens einer Kopie der geänderten Version in dem ersten Zwischenspeicher bis die erste Kopie der geänderten Version oder ein Versionsnachfolger von ihr dauerhaft gespeichert ist.

21. Verfahren gemäß Anspruch 19, ferner aufweisend den Schritt:

Freigeben der Ressource an dem ersten Knoten (306) als Reaktion auf den Hinweis, dass ein anderer Knoten der Mehrzahl von Knoten die Version der Ressource dauerhaft gespeichert hat, die mindestens so neu wie die geänderte Version ist.

22. Computerlesbares Medium, das eine Folge oder mehrere Folgen von Befehlen für die Übertragung einer Ressource von einem ersten Zwischenspeicher zu einem zweiten Zwischenspeicher enthält,

wobei die Ausführung der einen Folge oder der mehreren Folgen von Befehlen mittels eines Prozessors oder mehrerer Prozessoren den einen Prozessor oder die mehreren Prozessoren veranlasst, die Schritte des in irgendeinem der Ansprüche 1-21 geschilderten Verfahrens durchzuführen.

23. System zum Übertragen einer Ressource, welches System aufweist:

einen ersten Knoten, der einen ersten Zwischenspeicher aufweist, der kommunikativ mit einem zweiten Zwischenspeicher aus einem oder mehreren anderen Zwischenspeichern, die in einem oder mehreren anderen Knoten enthalten sind, gekoppelt ist;
wobei der erste Knoten eingerichtet ist, eine erste Kopie der Ressource in dem ersten Zwischenspeicher beizubehalten, während des Übertragens einer zweiten Kopie der Ressource von dem ersten Zwischenspeicher zu dem zweiten Zwischenspeicher, ohne die Ressource von dem ersten Zwischenspeicher vorher dauerhaft in einen persistenten Speicher (655) zu speichern; und
wobei der erste Knoten eingerichtet ist, mindestens eine Kopie der Ressource in dem ersten Zwischenspeicher beizubehalten bis die erste Kopie der Ressource oder ein Versionsnachfolger von ihr dauerhaft gespeichert ist.

24. System gemäß Anspruch 23, wobei der erste Knoten ein erster Datenbank-Server (A, B, C, D) ist und mindestens einer der ein oder mehreren anderen Knoten ein zweiter Datenbank-Server (A, B, C, D) ist, der den zweiten Zwischenspeicher aufweist.

25. System gemäß Anspruch 23, wobei

der erste Knoten eingerichtet ist es zu ermöglichen, dass die erste Kopie der Ressource in dem ersten Zwischenspeicher geändert wird, bevor die zweite Kopie zu dem zweiten Zwischenspeicher übertragen wird; und
der erste Knoten eingerichtet ist, es zu verhindern, dass die erste Kopie der Ressource verändert wird, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen worden ist.

26. System gemäß Anspruch 23, wobei

der erste Knoten eingerichtet ist, eine Anforderung der Genehmigung für das Freigeben der ersten Kopie nach dem Übertragen der zweiten Kopie zu dem zweiten Zwischenspeicher an den Hauptknoten (D) zu senden; und
der erste Knoten eingerichtet ist, eine Nachricht von dem Hauptknoten (D) zu empfangen, die anzeigt, dass die erste Kopie freigegeben werden kann, nachdem der Hauptknoten (D) es als Reaktion auf die Anforderung veranlasst, dass die erste Kopie oder einen Versionsnachfolger von ihr dauerhaft gespeichert wird.

27. System gemäß Anspruch 26, wobei

der erste Knoten einen Sendeprozess aufweist, der eingerichtet ist, eine Anforderung der Genehmigung für das Freigeben der ersten Kopie zu senden; und
ein anderer Prozess als der Sendeprozess einen Versionsnachfolger der ersten Kopie der Ressource speichert.

28. System gemäß Anspruch 23, wobei der erste Knoten eingerichtet ist, mindestens eine Kopie der Ressource in dem ersten Zwischenspeicher beizubehalten, durch:

Bestimmen, ob eine dauerhaft gespeicherte Kopie der Ressource neuer ist als die erste Kopie, bevor versucht wird, die erste Kopie dauerhaft zu speichern;
Freigeben der ersten Kopie ohne dauerhaftes Speichern der ersten Kopie, falls die dauerhaft gespeicherte Kopie neuer als die erste Kopie ist; und
dauerhaftes Speichern der ersten Kopie, falls die dauerhaft gespeicherte Kopie nicht neuer als die erste Kopie ist.

29. System gemäß Anspruch 25, wobei der erste Knoten eingerichtet ist, eine Änderungsgenehmigung von einem Sendeprozess, der dem ersten Zwischenspeicher zugeordnet ist (204), zu einem Empfangsprozess, der dem zweiten Zwischenspeicher zugeordnet ist, zusammen mit der zweiten Kopie der Ressource zu übertragen.

30. System gemäß Anspruch 29, wobei:

Genehmigungen für das Zugreifen auf die Ressource mittels eines Hauptservers (D) verwaltet werden; und
der erste Knoten eingerichtet ist, die Änderungsgenehmigung zu dem Empfangsprozess zu übertragen, bevor der erste Knoten die Bestätigung von dem Hauptserver (D) für die Übertragung der Änderungsgenehmigung zu dem Empfangsprozess empfängt.

31. System gemäß Anspruch 23, wobei:

der erste Knoten eingerichtet ist, eine Änderungsgenehmigung von einem Sendeprozess,

der dem ersten Zwischenspeicher zugeordnet ist, an einen Empfangsprozess, der dem zweiten Zwischenspeicher zugeordnet ist, zusammen mit der zweiten Kopie der Ressource zu übertragen;
Genehmigungen für das Zugreifen auf die Ressource mittels eines Hauptservers (D) verwaltet werden; und
das Übertragen der Änderungsgenehmigung zu dem Empfangsprozess durchgeführt wird, bevor der Hauptserver (D) die Bestätigung für die Übertragung der Änderungsgenehmigung zu dem Empfangsprozess empfängt.

32. System gemäß Anspruch 23, wobei:

der erste Knoten einen Sendeprozess aufweist, der dem ersten Zwischenspeicher zugeordnet ist, wobei der Sendeprozess eingerichtet ist, eine Nachricht von einem Hauptknoten (D) zu empfangen, der eine Anforderung für die Ressource von einem Empfangsprozess, der dem zweiten Zwischenspeicher zugeordnet ist, empfängt; und
der Sendeprozess die zweite Kopie zu dem Empfangsprozess als Reaktion auf die Nachricht von dem Hauptserver (D) überträgt.

33. Verfahren gemäß Anspruch 23, wobei:

der erste Knoten einen Sendeprozess aufweist, der eingerichtet ist, eine Sperre von einem Sperrmanager anzufordern, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen ist, wobei die Sperre die Genehmigung zum Schreiben der Ressource auf Platte (655) erteilt, aber nicht die Genehmigung zur Änderung der Ressource; und
der Sperrmanager einen Prozess auswählt, der eine Version der Ressource aufweist, die mindestens so neu wie die erste Kopie ist und dem ausgewählten Prozess die Sperre erteilt, um den ausgewählten Prozess zu veranlassen, die Version der Ressource auf Platte (655) zu schreiben.

34. System gemäß Anspruch 23, wobei der Sperrmanager als Reaktion auf das Schreiben der Version der Ressource auf Platte (655) veranlasst, dass alle Versionen der Ressource, die älter als die Version sind, freigegeben werden.

35. System gemäß Anspruch 23, ferner aufweisend:

einen Hauptknoten (D), der eingerichtet ist zu bestimmen, nach einem Ausfall eines Zwischenspeichers, der eine verschmutzte Kopie der Ressource enthält, ob ein ausgefallener Zwischenspeicher die letzte Version der Ressource enthalten hat; und
wobei, falls der ausgefallene Zwischenspeicher die letzte Version der Ressource enthalten hat, der Hauptknoten (D) eingerichtet ist zu veranlassen, dass eine letzte frühere Abbild-Kopie der Ressource auf Platte (655) geschrieben wird, zu veranlassen, dass alle vorherigen früheren Abbild-Kopien der Ressource freigegeben werden, und zu veranlassen, dass ein Wiederherstellungsprotokoll des ausgefallenen Zwischenspeichers verwendet wird, um die letzte Version der Ressource zu rekonstruieren.

36. System gemäß Anspruch 35, wobei der Hauptknoten (D) eingerichtet ist, falls der ausgefallene Zwischenspeicher nicht die letzte Version der Ressource enthalten hat, zu veranlassen, dass die letzte Version der Ressource auf Platte (655) geschrieben wird und zu veranlassen, dass alle früheren Abbild-Kopien der Ressource freigegeben werden.

37. Verfahren gemäß Anspruch 23, ferner aufweisend:

einen Hauptknoten (D), der eingerichtet ist, nach einem Ausfall einer Mehrzahl von Zwischenspeichern, die verschmutzte Versionen der Ressource enthalten, zu bestimmen, ob einer der ausgefallenen Zwischenspeicher die letzte Version der Ressource enthalten hat und, falls irgendeiner der ausgefallenen Zwischenspeicher die letzte Version der Ressource enthalten hat, die Wiederherstellungsprotokolle der ausgefallenen Zwischenspeicher zusammenzuführen und zu verwenden, um die letzte Version der Ressource zu rekonstruieren.

38. System gemäß Anspruch 23, wobei der erste Zwischenspeicher und der zweite Zwischenspeicher zu einer Mehrzahl von Zwischenspeichern gehören, wobei ein separates Wiederherstellungsprotokoll für jeden Zwischenspeicher der Mehrzahl von Zwischenspeichern verwaltet wird und das System ferner aufweist:

einen Hauptknoten (D), der eingerichtet ist, einen ausgefallenen Zwischenspeicher der Mehrzahl von Zwischenspeichern wiederherzustellen, basierend auf dem Wiederherstellungsprotokoll, das dem ausgefallenen Zwischenspeicher zugeordnet ist, ohne die separaten Wiederherstellungsprotokolle der anderen Zwischenspeicher der Mehrzahl von Zwischenspeichern zu überprüfen.

39. System gemäß Anspruch 38, wobei jeder Zwischenspeicher der Mehrzahl von Zwischenspei-

chern ein Zwischenspeicher ist, der von einem separaten Datenbank-Server einer Mehrzahl von Datenbank-Servern (A, B, C, D) verwaltet wird.

40. System gemäß Anspruch 38, wobei

der erste Knoten eingerichtet ist es zu ermöglichen, dass die erste Kopie der Ressource in dem ersten Zwischenspeicher verändert wird, bevor die zweite Kopie zu dem zweiten Zwischenspeicher übertragen wird; und
der erste Knoten eingerichtet ist zu verhindern, dass die erste Kopie der Ressource verändert wird, nachdem die zweite Kopie zu dem zweiten Zwischenspeicher übertragen worden ist.

41. Verfahren gemäß Anspruch 23, wobei
der erste Knoten eingerichtet ist:

die Ressource in dem ersten Zwischenspeicher des ersten Knotens zu modifizieren, so dass eine geänderte Version der Ressource erzeugt wird;
einen Kontrollpunkts für den ersten Knoten zu verwalten, der anzeigt, wo mit der Arbeit begonnen werden muss, wenn der erste Knoten ausfällt;
eine ersten Kopie der geänderten Version in dem ersten Zwischenspeicher während des Übertragens einer zweiten Kopie der geänderten Version von dem ersten Zwischenspeicher zu dem zweiten Zwischenspeicher des zweiten Knotens beizubehalten, ohne die geänderte Version von dem ersten Zwischenspeicher vorher dauerhaft in einen persistenten Speicher zu speichern; und
den Kontrollpunkts als Reaktion auf den Hinweis, das ein anderer Knoten der Mehrzahl von Knoten (A, B, C, D) eine Version der Ressource dauerhaft gespeichert hat, die mindestens so neu wie die geänderte Version ist, vorzuversetzen.

42. System gemäß Anspruch 41, wobei der erste Knoten ferner eingerichtet ist:

mindestens eine Kopie der geänderten Version in dem ersten Zwischenspeicher beizubehalten, bis die erste Kopie der geänderten Version oder ein Versionsnachfolger von ihr dauerhaft gespeichert ist.

43. System gemäß Anspruch 41, wobei der erste Knoten ferner eingerichtet ist:

die Ressource an dem ersten Knoten als Reaktion auf den Hinweis, dass ein anderer Knoten der Mehrzahl von Knoten (A, B, C, D) die Version der Ressource dauerhaft gespeichert hat, die mindestens so neu wie die geänderte Version ist, freizugeben.

44. System gemäß Anspruch 23,
wobei der erste Knoten eingerichtet ist zu verhindern, dass die erste Kopie in dem ersten Zwischenspeicher ersetzt wird, bis die erste Kopie oder ein Versionsnachfolger von ihr dauerhaft gespeichert ist.

Revendications

1. Procédé de transfert d'une ressource d'une première mémoire cache vers une seconde mémoire cache, le procédé comprenant les étapes consistant à:

conserver une première copie de la ressource dans ladite première mémoire cache tout en transférant une seconde copie de la ressource jusqu'à la première mémoire cache en direction de la seconde mémoire cache sans stocker de façon durable ladite ressource de ladite première mémoire cache vers une mémoire permanente (655); et
conserver au moins une copie de la ressource dans ladite première mémoire cache jusqu'à ce que ladite première copie de la ressource ou de son successeur soit stockée durablement.

2. Procédé selon la revendication 1, dans lequel ladite première mémoire cache est une mémoire cache conservée par un premier serveur de base de données (A, B, C, D) et ladite deuxième mémoire cache est une mémoire cache conservée par un deuxième serveur de base de données (A, B, C, D).

3. Procédé selon la revendication 1, comprenant en outre les étapes consistant à:

autoriser une modification de ladite ressource dans ladite mémoire cache avant le transfert de ladite seconde copie dans ladite seconde mémoire cache; et
empêcher une modification de ladite première copie de ladite ressource après le transfert de ladite seconde copie dans ladite seconde mémoire cache.

4. Procédé selon la revendication 1, comprenant en outre les étapes consistant à, après le transfert de ladite seconde copie dans ladite seconde mémoire cache, émettre une demande d'autorisation pour la libération de ladite première copie;

en réponse à ladite demande, provoquer le

stockage durable (302) de ladite première copie ou un successeur de cette copie; et
en réponse au stockage durable dudit successeur, émettre un message qui indique que ladite première copie peut être libérée (304).

5. Procédé selon la revendication 4, selon lequel:

l'étape consistant à envoyer une demande d'autorisation de libération de ladite première copie est exécutée au moyen d'un processus d'émission; et
l'étape consistant à provoquer le stockage durable de ladite première copie ou d'un successeur de cette dernière inclut l'étape consistant à amener un procédé autre que le processus d'émission à stocker un successeur à ladite première copie de ladite ressource (302).

6. Procédé selon la revendication 1, selon lequel l'étape de conservation d'au moins une copie de la ressource dans ladite première mémoire comprend les étapes consistant à:

avant d'essayer de stocker de façon durable ladite première copie, déterminer si une copie stockée de façon durable de ladite ressource est plus récente que ladite première copie;
si ladite copie stockée de façon durable est plus récente que ladite première copie, libérer alors ladite première copie sans stocker durablement ladite première copie; et
si ladite copie stockée durablement n'est pas plus récente que ladite première copie, alors stocker durablement ladite première copie.

7. Procédé selon la revendication 3, comprenant en outre l'étape consistant à transférer une autorisation de modification à partir d'un processus d'émission associé à la première mémoire cache (204) pour un processus de réception associé à la seconde mémoire cache conjointement avec ladite seconde copie de ladite ressource.

8. Procédé selon la revendication 7, selon lequel:

les autorisations d'accès à ladite ressource sont gérées par un maître D, et
l'étape de transfert de ladite autorisation de modification au processus de réception est exécutée avant la réception d'un accusé de réception de la part dudit maître (D) pour le transfert de ladite autorisation de modification pour ledit processus de réception (204).

9. Procédé selon la revendication 1, comprenant en outre les étapes consistant à:

transférer une autorisation de modification faisant passer d'un processus d'émission associé à ladite première mémoire cache à un processus de réception (204) associé à ladite seconde mémoire cache conjointement avec ladite seconde copie de la ressource;
selon lequel des autorisations d'accès à ladite ressource sont gérées par un maître (D); et
selon lequel l'étape de transfert de ladite autorisation de modification vers ledit processus de réception est exécutée avant que ledit maître (D) ne reçoive l'accusé de réception du transfert de ladite autorisation de modification audit processus de réception (204).

10. Procédé selon la revendication 1, comprenant en outre les étapes:

un processus de réception associé à ladite seconde mémoire cache envoyant une demande pour ladite ressource à un maître (D) de ladite ressource;
en réponse à ladite demande provenant dudit procédé de réception, envoi d'un message par ledit maître (D) de ladite ressource à un processus d'émission associé à ladite première mémoire cache; et
transfert de ladite seconde copie par ledit processus d'émission audit processus de réception en réponse audit message provenant dudit maître (D).

11. Procédé selon la revendication 1, comprenant en outre l'exécution des étapes indiquées ci-après après l'étape de transfert de ladite seconde copie à ladite seconde mémoire cache:

un processus d'émission associé à la première mémoire cache demandant un accès (300) à partir du gestionnaire d'accès, ledit accès accordant l'autorisation d'écrire ladite ressource sur le disque, mais pas l'autorisation de modifier ladite ressource;
ledit gestionnaire d'accès sélectionnant un procédé qui possède une version de ladite ressource qui est au moins aussi récente que ladite première copie;
ledit gestionnaire d'accès accordant ledit accès audit processus sélectionné; et
ledit procédé sélectionné écrivant ladite version de ladite ressource sur le disque.

12. Procédé selon la revendication 11, comprenant en outre l'étape consistant en ce qu'en réponse au fait que ladite version de ladite ressource est écrite sur le disque (302), ledit gestionnaire d'accès provoque la libération (306) de toutes les versions de ladite ressource, qui sont plus anciennes que ladite ver-

sion.

13. Procédé selon la revendication 1, comprenant en outre les étapes consistant à, après une défaillance d'une mémoire cache qui conserve une copie usagée de ladite ressource:

déterminer si la mémoire cache défaillante a conservé la version la plus récente de la ressource (402);
si la mémoire cache défaillante a conservé la version la plus récente de la ressource, alors
écrire la dernière image passée de la ressource sur un disque (408);
libérer toutes les images passées précédentes de la ressource (410); et
appliquer un journal de récupération de ladite mémoire cache défaillante pour reconstituer la version la plus récente de la ressource (412).

14. Procédé selon la revendication 13, comprenant en outre les étapes consistant à:

si la mémoire défaillante n'a pas conservé la version la plus récente de la ressource, alors
écrire la version la plus récente de la ressource sur un disque (404); et
libérer toutes les images passées de la ressource (406).

15. Procédé selon la revendication 1, comprenant en outre les étapes consistant à, après une défaillance d'une pluralité de mémoires cache qui conservent des versions usagées de ladite ressource:

déterminer si l'une quelconque des mémoires cache défaillante a conservé la version la plus récente de la ressource (400); et
si l'une quelconque des mémoires cache défaillantes a conserver la version la plus récente de la ressource, alors
fusionner et appliquer les journaux de récupération desdites mémoires cache défaillantes pour reconstituer la version la plus récente de la ressource.

16. Procédé selon la revendication 1, selon lequel la seconde copie de la ressource est une version usagée de la ressource, dans laquelle la première mémoire cache et la seconde mémoire cache font partie d'une pluralité de mémoires cache et comprenant en outre les étapes consistant à:

conserver des journaux de récupération séparés pour chaque mémoire cache de ladite pluralité de mémoires cache; et
lorsqu'une mémoire cache de ladite pluralité de mémoires cache est défaillante, récupérer la mémoire cache défaillante sur la base du journal de récupération associé à ladite mémoire cache défaillante sans inspection des journaux de récupération séparés des autres mémoire cache de ladite pluralité de mémoires cache.

17. Procédé selon la revendication 16, selon lequel chaque mémoire cache de la pluralité de mémoires cache est une mémoire cache conservée par un serveur de base de donnée séparé (A, B, C, D) d'une pluralité de serveurs de base de données (A, B, C, D).

18. Procédé selon la revendication 16, selon lequel:

le procédé comprend en outre les étapes consistant à:

autoriser une modification de ladite première copie de ladite ressource dans ladite première mémoire cache avant le transfert de ladite seconde copie à ladite seconde mémoire cache; et
empêcher une modification de ladite première copie de ladite ressource après le transfert de ladite seconde copie à ladite seconde mémoire cache.

19. Procédé selon la revendication 1, selon lequel la première mémoire cache réside en un premier noeud d'une pluralité de noeuds (A, B, C, D), et la seconde mémoire cache réside dans un second noeud de ladite pluralité de noeuds (A, B, C, D) et comprenant en outre les étapes consistant à:

modifier ladite ressource dans ladite première mémoire cache du premier noeud parmi une pluralité de noeuds (A, B, C, D) pour créer une version modifiée de ladite ressource;
maintenir un point de contrôle dudit premier noeud qui indique où commencer le travail lorsque ledit premier noeud est défaillant;
conserver une première copie de ladite version modifiée dans ladite première mémoire cache tout en transférant une seconde copie de ladite version modifiée depuis ladite première mémoire cache vers ladite seconde mémoire cache du second noeud de la pluralité de noeuds (A, B, C, D) sans stocker tout d'abord de façon durable ladite version modifiée depuis ladite première mémoire cache vers une mémoire permanente (655); et
en réponse à une indication du fait qu'un autre noeud parmi ladite pluralité de noeuds (A, B, C, D) a stocké durablement une version de ladite ressource qui est au moins aussi récente que ladite version modifiée, avancer le point de contrôle.

20. Procédé selon la revendication 19, comprenant en outre l'étape consistant à:

conserver au moins une copie de ladite version modifiée dans ladite première mémoire cache jusqu'à ce que ladite première copie de ladite version modifiée ou un successeur de cette copie soit stockée durablement.

21. Procédé selon la revendication 19, comprenant en outre l'étape consistant à:

en réponse à l'indication du fait qu'un autre noeud de ladite pluralité de noeuds a stocké durablement la version de ladite ressource, qui est au moins aussi récente que ladite version modifiée, libérer ladite ressource dans le premier noeud (306).

22. Support lisible par ordinateur portant une ou plusieurs séquences d'instructions pour transférer une ressource depuis une première mémoire cache vers une seconde mémoire cache, l'exécution de la une ou plusieurs séquences d'instructions par un processeur ou la pluralité de processeurs amène le processeur ou la pluralité de processeurs à exécuter les étapes du procédé indiqué dans l'une quelconque des revendications 1 à 21.

23. Système pour transférer une ressource, le système comprenant:

un premier noeud qui possède une première mémoire cache qui est couplée, de manière à communiquer à une seconde mémoire cache parmi une ou plusieurs autres mémoires cache, qui sont contenues dans un ou plusieurs autres noeuds;
dans lequel ledit premier noeud est configuré de manière à conserver une première copie de la ressource dans la première mémoire cache tout en transférant une seconde partie de la ressource depuis la première mémoire cache vers la seconde mémoire cache sans mémoriser tout d'abord de façon durable ladite ressource depuis ladite première mémoire cache en direction d'une mémoire permanente (655); et
dans lequel ledit premier noeud est configuré de manière à conserver au moins une copie de la ressource dans ladite première mémoire cache jusqu'à ce que ladite première copie de la ressource ou un successeur de cette copie soit stockée de façon durable.

24. Système selon la revendication 23, dans lequel ledit premier noeud est un premier serveur de base de données (A, B, C, D), et au moins un du ou desdits plusieurs autres noeuds est le second serveur de base de données (A, B, C, D) qui inclut ladite seconde mémoire cache.

25. Système selon la revendication 23, dans lequel

ledit premier noeud est configuré de manière à permettre une modification de ladite première copie de ladite ressource dans ladite première mémoire cache avant le transfert de ladite seconde copie à ladite seconde mémoire cache; et
ledit premier noeud est configuré de manière à empêcher que ladite première copie de ladite ressource soit modifiée après le transfert de ladite seconde copie à ladite seconde mémoire cache.

26. Système selon la revendication 23, dans lequel:

ledit premier noeud est configuré de manière à envoyer une demande à un noeud maître (D) pour permettre la libération de ladite première copie, après transfert de ladite seconde copie à ladite seconde mémoire cache; et
ledit premier noeud est configuré de manière à recevoir de la part dudit noeud maître (D) un message qui indique dans ladite première copie peut être libérée après que ledit premier noeud maître (D) a provoqué la mémorisation durable de ladite première copie ou d'un successeur de cette dernière, en réponse à ladite demande.

27. Système selon la revendication 26, dans lequel

ledit premier noeud comprend un procédé d'émission qui est conçu de manière à émettre la demande d'autorisation de libération de ladite première copie; et
un procédé autre que le procédé d'émission mémorise un successeur à ladite première copie de ladite ressource.

28. Système selon la revendication 23, dans lequel ledit premier noeud est conçu de manière à conserver au moins une copie de la ressource dans ladite première mémoire cache par:

avant d'essayer de mémoriser de façon durable ladite première copie, détermination du fait qu'une copie mémorisée de façon durable de ladite ressource est plus récente que ladite première copie;
si ladite copie mémorisée de façon durable est plus récente que ladite première copie, alors libération de ladite première copie sans mémorisation durable de ladite première copie; et

si ladite copie mémorisée de façon durable n'est pas plus récente que ladite première copie, alors mémorisation durable de ladite première copie.

29. Système selon la revendication 25, dans lequel ledit premier noeud est configuré de manière à transférer une autorisation de modification depuis un procédé d'émission associé à la première mémoire cache à un procédé de réception associé à la seconde mémoire cache, conjointement avec une seconde copie de ladite ressource.

30. Système selon la revendication 29, dans lequel:

les autorisations d'accès à ladite ressource sont gérées par un noeud maître (D); et ledit premier noeud est configuré de manière à transférer ladite autorisation de modification au procédé de réception avant que ledit premier noeud reçoit un accusé de réception de la part dudit noeud maître (D) pour le transfert de ladite autorisation de modification audit procédé de réception.

31. Système selon la revendication 23, dans lequel:

ledit premier noeud est configuré de manière à transférer une autorisation de modification depuis un procédé d'émission associé à ladite première mémoire cache à un procédé de réception associé à ladite seconde mémoire cache conjointement avec ladite seconde copie de ladite ressource;
des autorisations d'accès à ladite ressource sont gérées par un maître (D); et
le transfert de ladite autorisation de modification audit procédé de réception est exécuté avant que ledit maître (D) reçoive l'accusé de réception du transfert de ladite autorisation de modification audit procédé de réception.

32. Système selon la revendication 23, dans lequel:

ledit premier noeud inclut un procédé d'émission associé à ladite première mémoire cache, selon lequel procédé d'émission étant configuré de manière à recevoir un message de la part d'un noeud maître (D) qui reçoit une demande pour ladite ressource à partir d'un procédé de réception associé à ladite seconde mémoire cache; et
ledit procédé d'émission transfert ladite seconde copie audit procédé de réception en réponse audit message provenant dudit noeud maître (D).

33. Système selon la revendication 23, dans lequel:

ledit premier noeud inclut un procédé d'émission qui est configuré de manière à demander un accès à partir d'un gestionnaire d'accès après que ladite seconde copie a été transférée à ladite seconde mémoire cache, ledit accès octroyant une autorisation d'écrire ladite ressource sur ledit disque (655) mais pas l'autorisation de modifier ladite ressource; et
ledit gestionnaire d'accès sélectionne un procédé qui possède une version de ladite ressource, qui est au moins aussi récente que ladite première copie et octroie ledit accès audit procédé sélectionné pour amener ledit procédé sélectionner à écrire ladite version de ladite ressource sur le disque (655).

34. Système selon la revendication 23, dans lequel ledit gestionnaire d'accès provoque la libération de toutes les versions de ladite ressource qui sont plus anciennes que ladite version, en réponse au fait que ladite version de ladite ressource est écrite sur le disque (655).

35. Système selon la revendication 23, comprenant en outre:

un noeud maître (D) qui est configuré de manière à déterminer, après une défaillance d'une mémoire cache qui conserve une copie usagée de ladite ressource, si une mémoire cache en panne a conservé la version la plus récente de la ressource; et
dans lequel, si la mémoire cache défaillante a conservé la version la plus récente de la ressource, le noeud maître (D) est configuré de manière à provoquer l'écriture d'une image passée la plus récente de la ressource sur le disque (655) de manière à provoquer la libération de toutes les images passées antérieures de la ressource, et pour provoquer l'application d'un journal de récupération de ladite mémoire cache en panne pour reconstituer la version la plus récente de la ressource.

36. Système selon la revendication 35, dans lequel si la mémoire cache défaillante n'a pas conservé la version la plus récente de la ressource, le noeud maître (D) déclenche l'écriture de la version la plus récente de la ressource sur le disque (655) et déclenche la libération de toutes les images passées de la ressource.

37. Système selon la revendication 23, comprenant en outre:

un noeud maître (D) qui est configuré de manière à déterminer, après une défaillance d'une pluralité de mémoires cache qui conservent

des versions usagées de ladite ressource, si l'une quelconque des mémoires cache défaillantes a conservé la version la plus récente de la ressource et, si l'une quelconque des mémoires défaillantes a conservé la version la plus récente de la ressource, alors fusionner et appliquer les journaux de récupération desdites mémoires cache défaillantes pour reconstituer la version la plus récente de la ressource.

38. Système selon la revendication 23, dans lequel la première mémoire cache et la seconde mémoire cache font partie d'une pluralité de mémoires cache et dans lequel ledit journal de récupération séparé est conservé pour chaque mémoire cache de ladite pluralité de mémoires cache, et ledit système comporte en outre:

un noeud maître (D), qui est configuré de manière à récupérer une mémoire cache défaillante parmi ladite pluralité de mémoires cache sur la base du journal de récupération associé à ladite mémoire cache défaillante sans inspection des journaux de récupération séparés des autres mémoires cache de ladite pluralité de mémoires cache.

39. Système selon la revendication 38, dans lequel chaque mémoire cache de ladite pluralité de mémoires cache est une mémoire cache conservée par un serveur de base de données séparé parmi une pluralité de serveurs de base de données (A, B, C, D).

40. Système selon la revendication 38, dans lequel:

le premier noeud est configuré de manière à permettre une modification de ladite première copie de ladite ressource dans ladite première mémoire cache avant le transfert de ladite seconde copie à ladite seconde mémoire cache; et
le premier noeud est configuré de manière à empêcher que ladite première copie de ladite ressource soit modifiée après le transfert de ladite seconde copie à ladite seconde mémoire cache.

41. Système selon la revendication 23, dans lequel ledit premier noeud est configuré de manière à:

modifier ladite ressource dans la première mémoire cache dudit premier noeud pour créer une version modifiée de ladite ressource;
conserver un point de contrôle pour ledit premier mode qui indique où le travail doit commencer lorsque ledit premier noeud est défaillant;
conserver une première copie de ladite version modifiée dans ladite première mémoire cache tout en conservant une seconde copie de ladite version modifiée depuis la première mémoire cache dans la seconde mémoire cache sans mémoriser tout d'abord de façon durable ladite version modifiée depuis ladite première mémoire cache vers une mémoire permanente; et en réponse à une indication qu'un autre noeud de ladite pluralité de noeuds (A, B, C, D) a stocké de façon durable une version de ladite ressource qui est au moins aussi récente que ladite version modifiée, faire avancer le point de contrôle.

42. Système selon la revendication 41, dans lequel le premier noeud est en outre configuré de manière à:

conserver au moins une copie de ladite version modifiée dans ladite première mémoire cache jusqu'à ce que ladite première copie de ladite version modifiée et d'un successeur de cette version soit stocké de façon durable.

43. Système selon la revendication 41, dans lequel le premier noeud est en outre configuré pour:

en réponse à l'indication qu'un autre noeud parmi ladite pluralité de noeuds (A, B, C, D) a stocké de façon durable la version de ladite ressource qui est au moins aussi récente que ladite version modifiée, libérer ladite ressource au niveau du premier noeud.

44. Système selon la revendication 23, dans lequel ledit premier noeud est configuré de manière à empêcher que ladite première copie soit remplacée dans ladite première mémoire cache jusqu'à ce que ladite première copie de la ressource ou un successeur de cette copie soit stocké de façon durable.

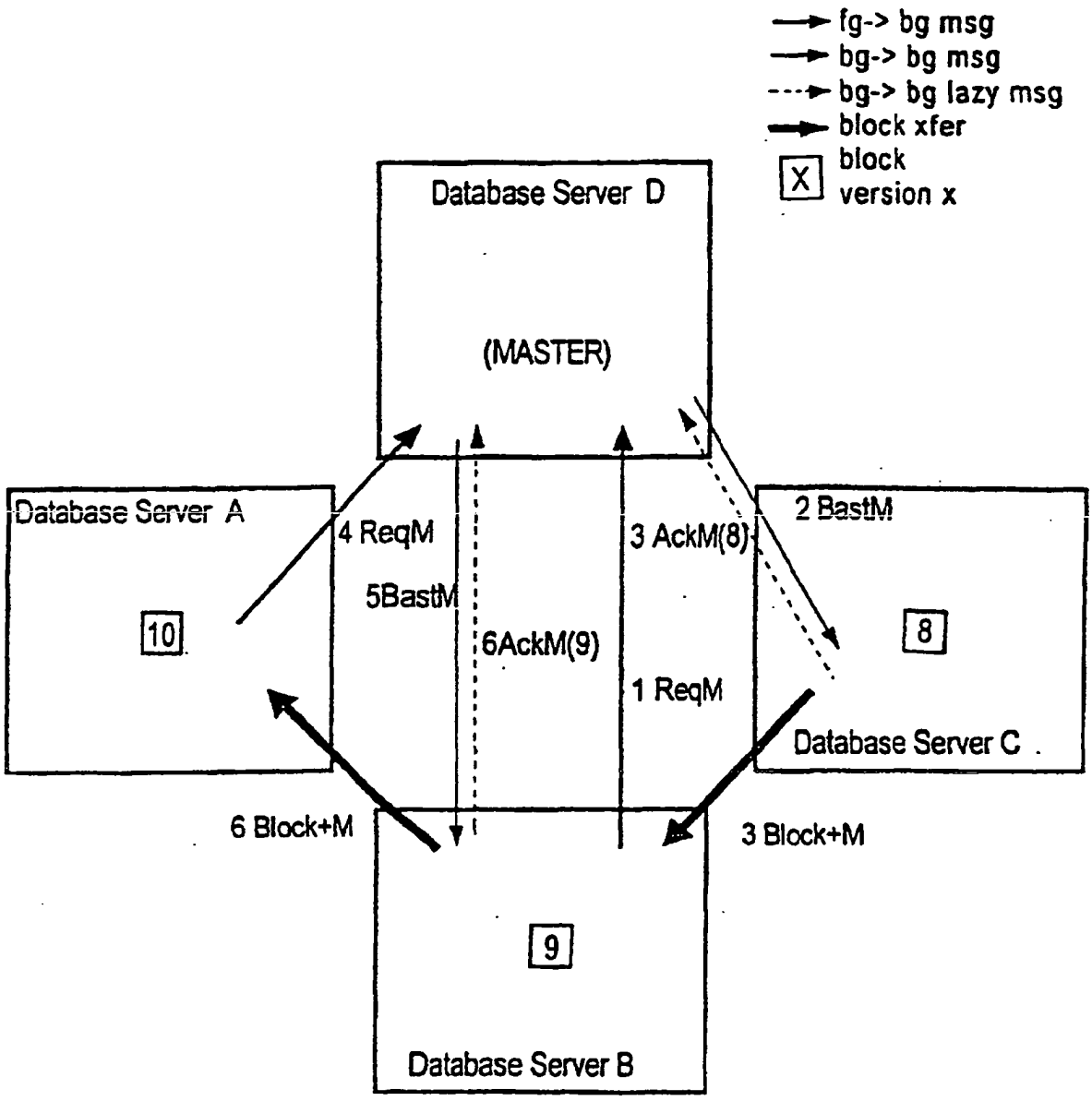


Fig. 1

Fig. 2

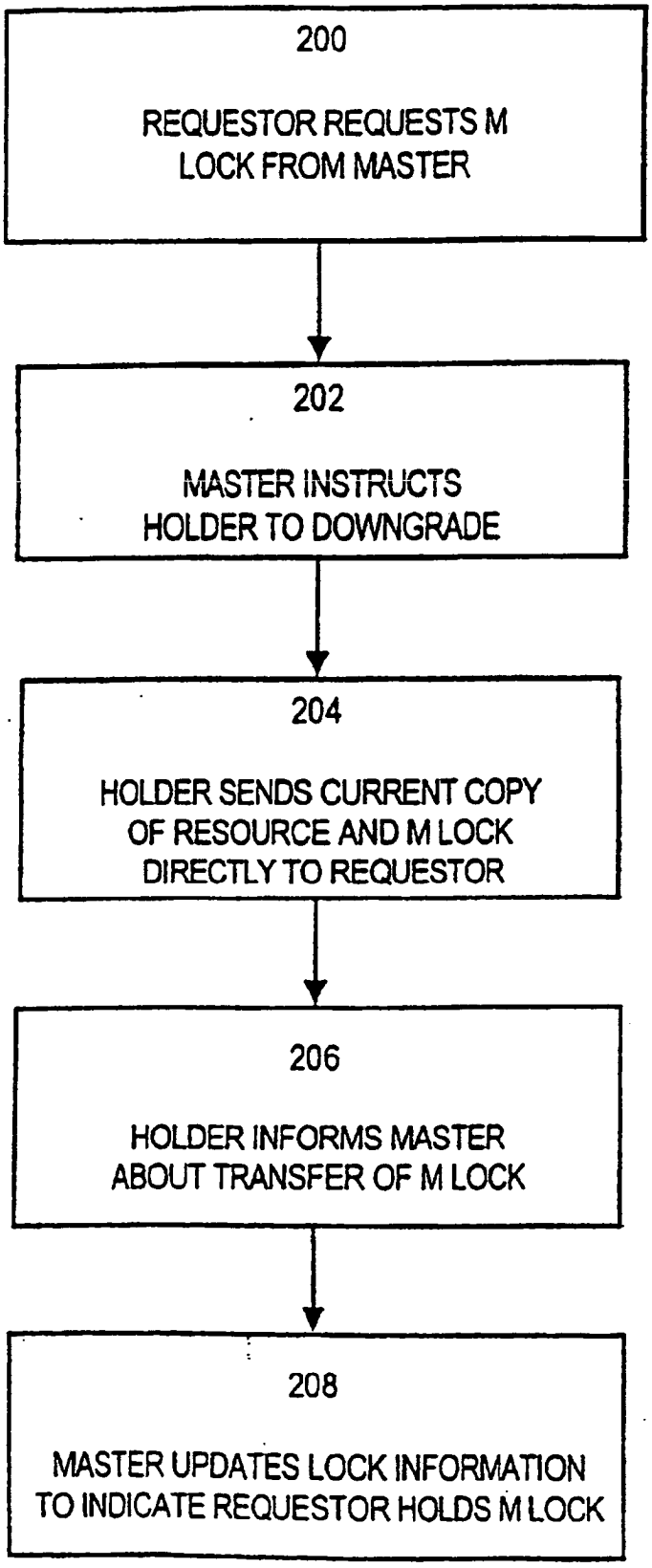


Fig. 3

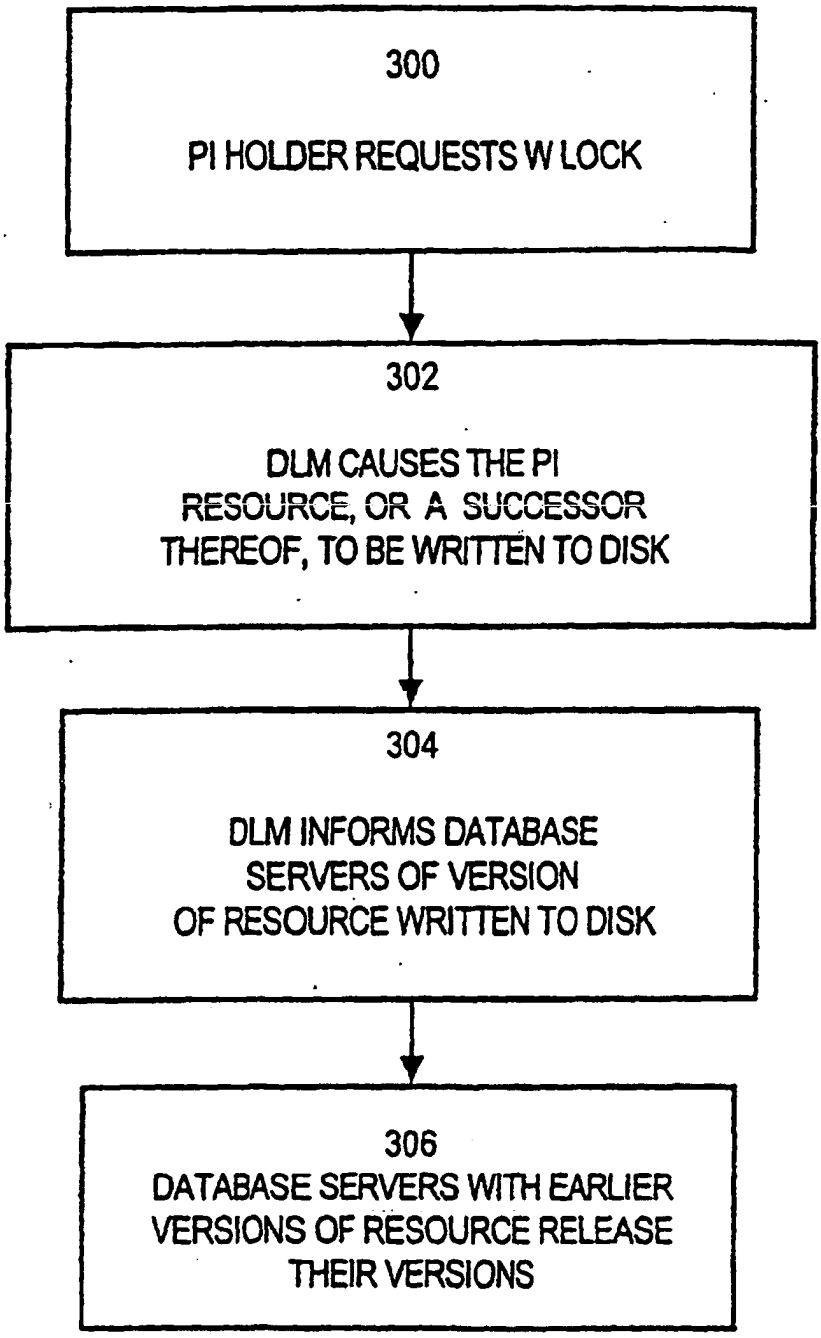
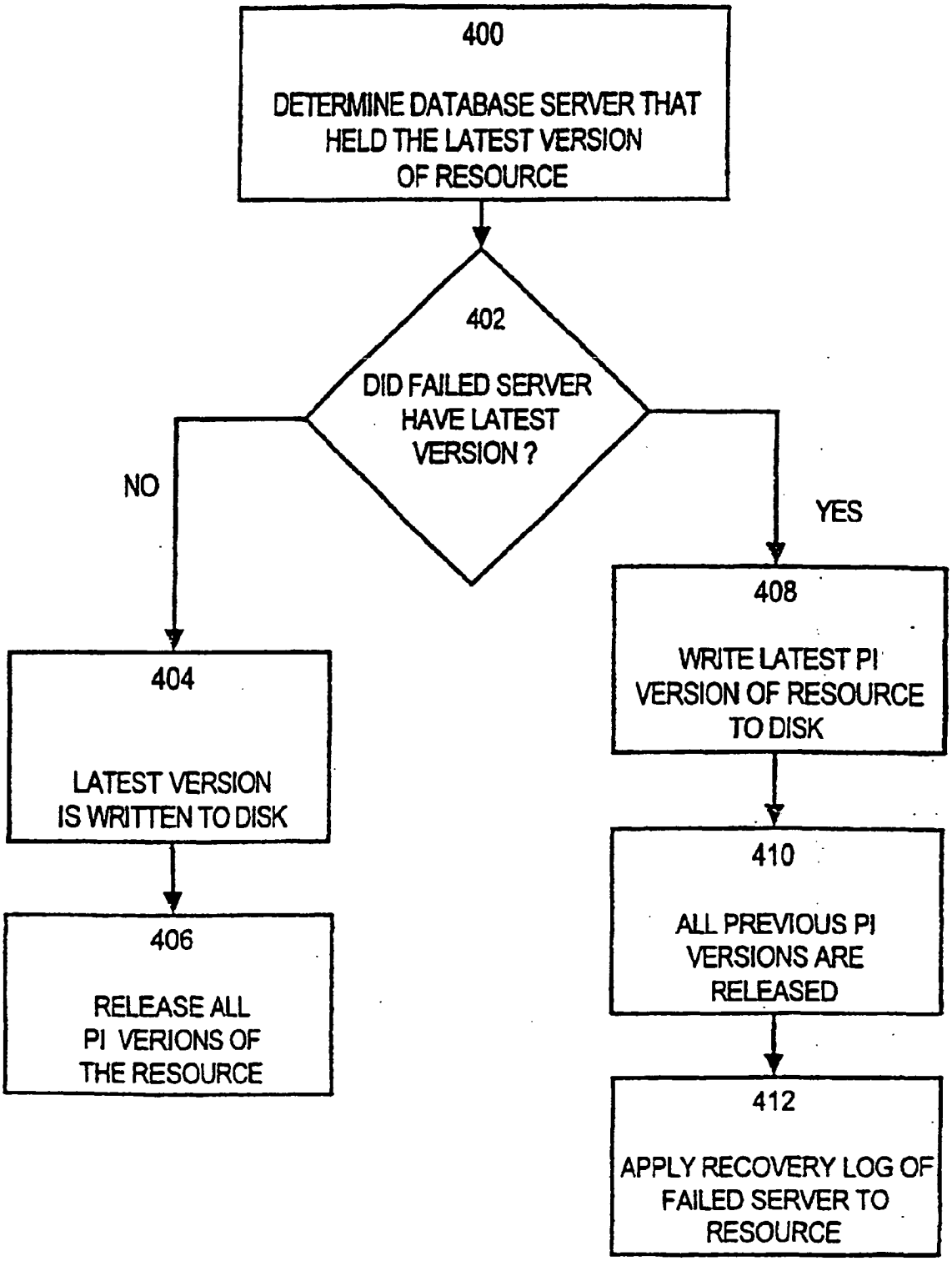


Fig. 4



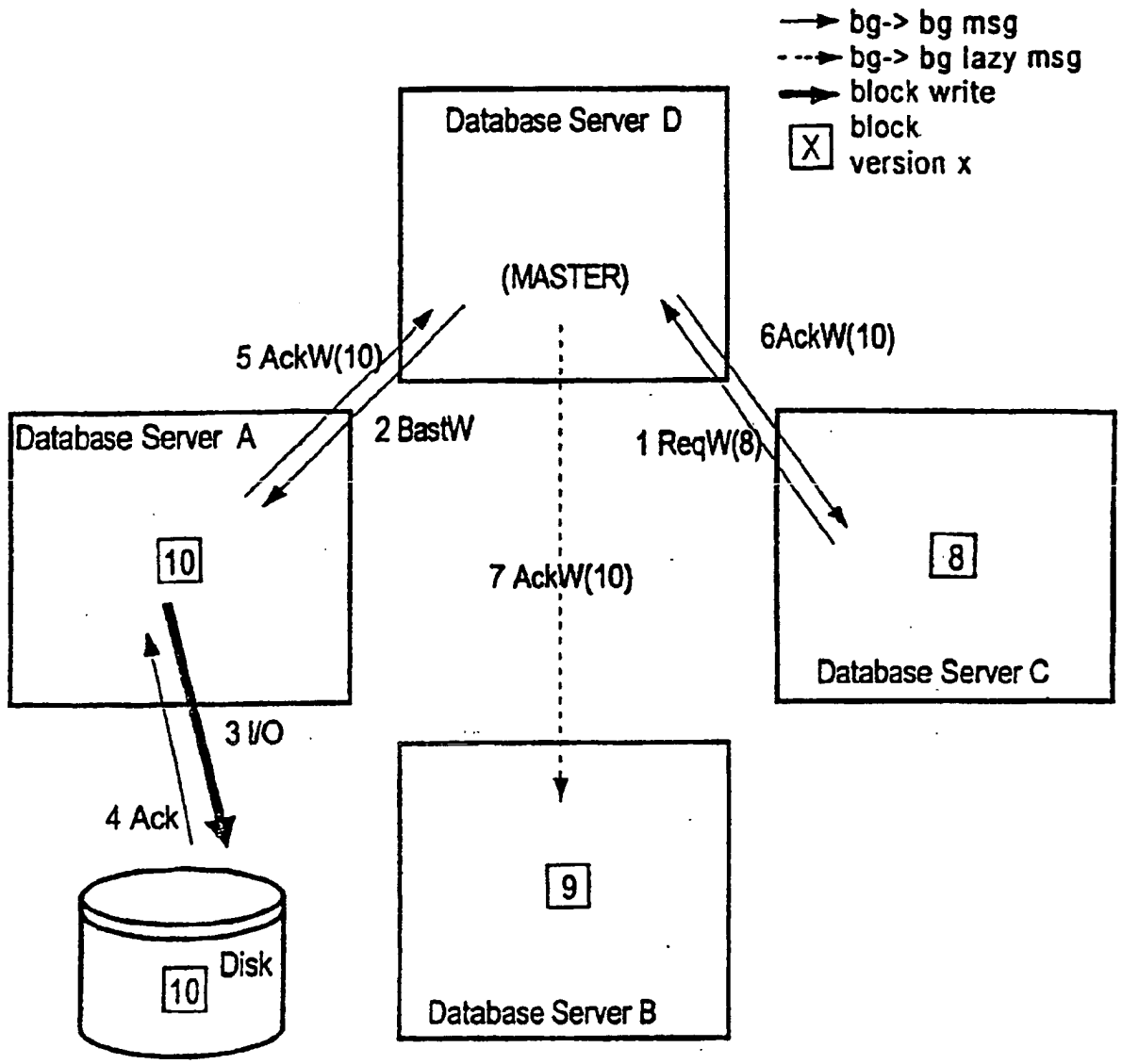


Fig. 5

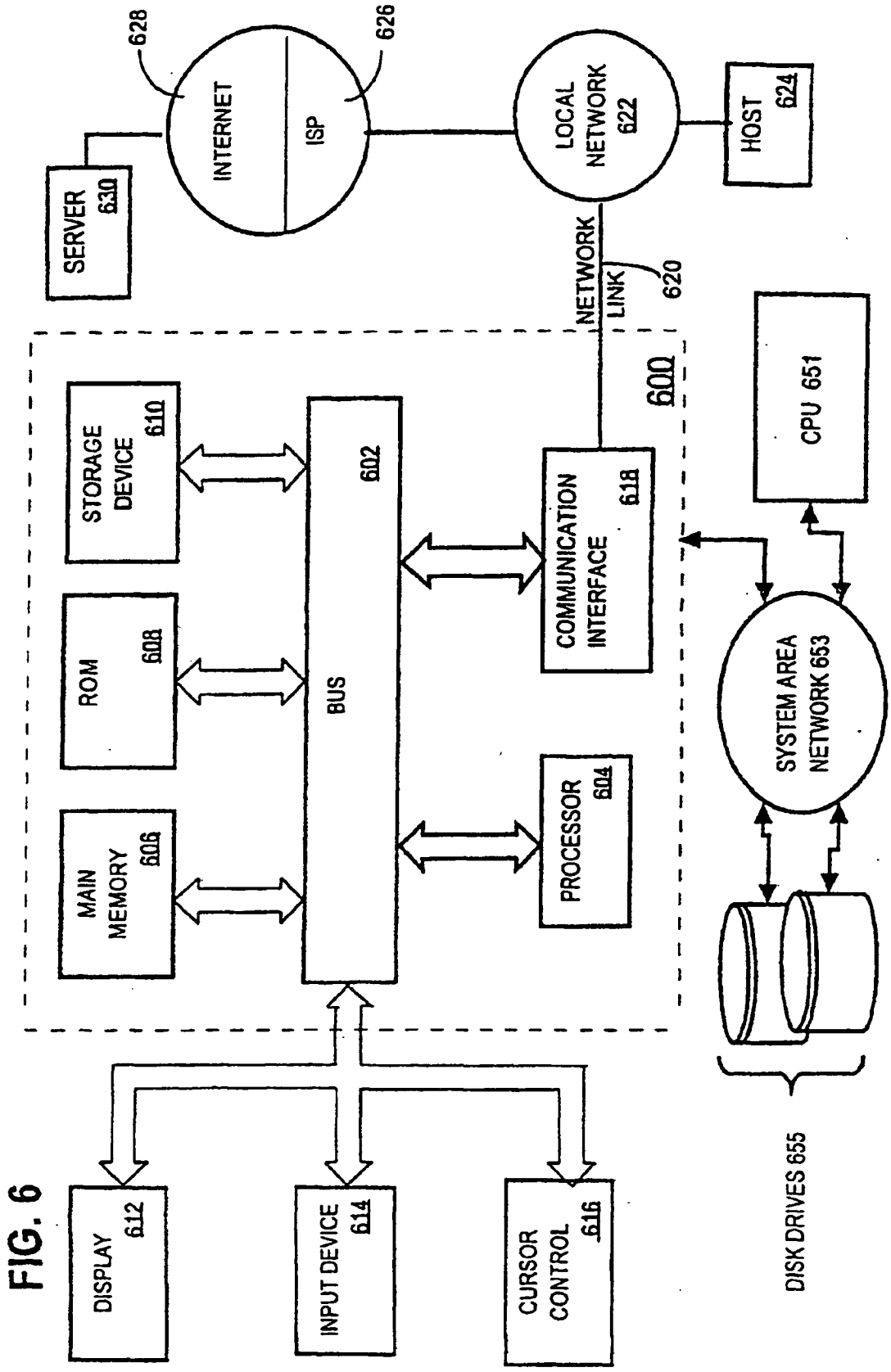


FIG. 6